Accelerating Host Networking in the Cloud

6.829: Computer Networks – 11/6/2018

Daniel Firestone

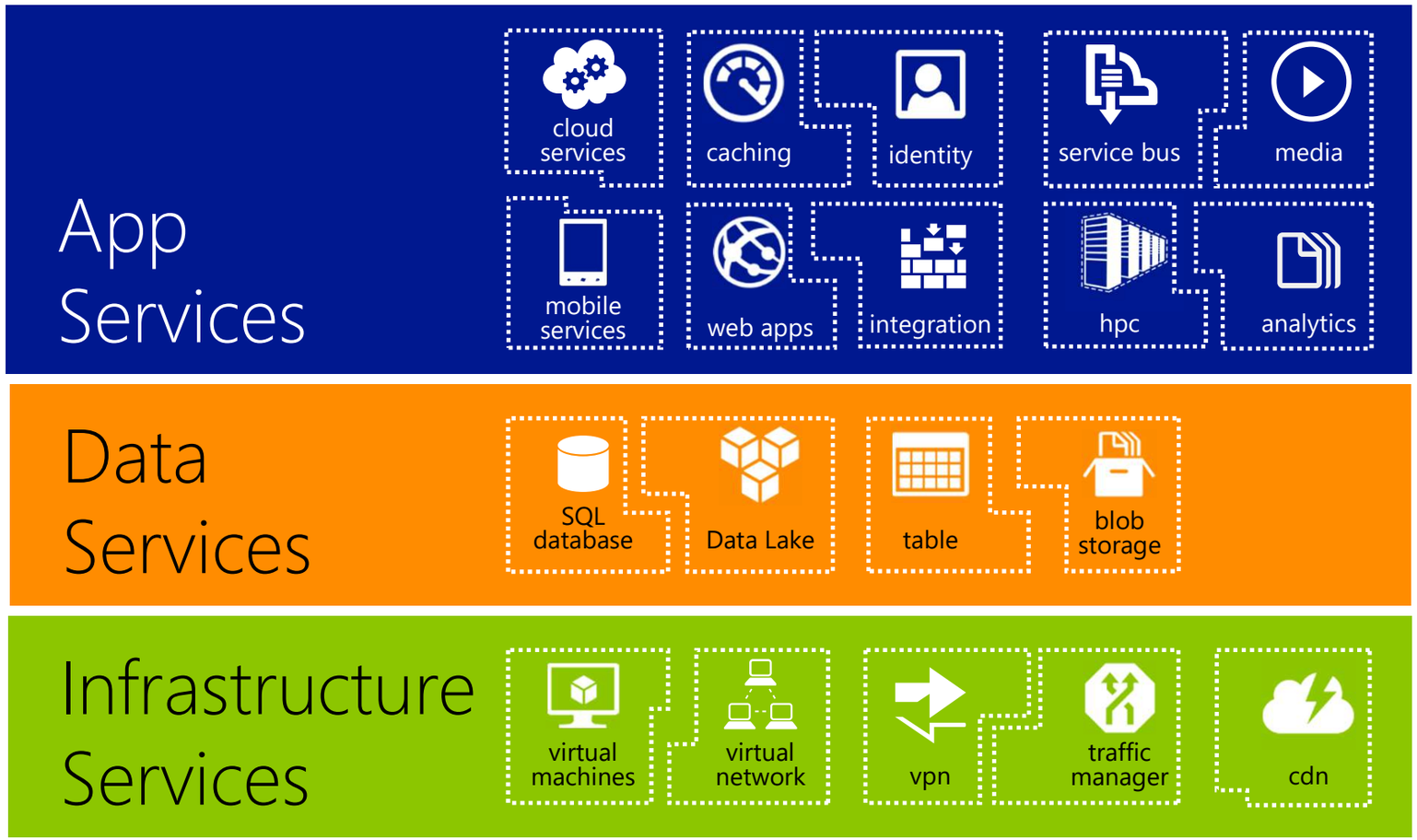
Tech Lead and Group Manager, Azure Host Networking Team



Overview

- Cloud and Scale
- Why SDN in the Cloud?
- Making SDN scale on the host
- VFP – Azure's platform for host SDN
- Accelerating host SDN with programmable hardware
- Azure Accelerated Networking - Results
- Conclusion and Future

Microsoft Azure

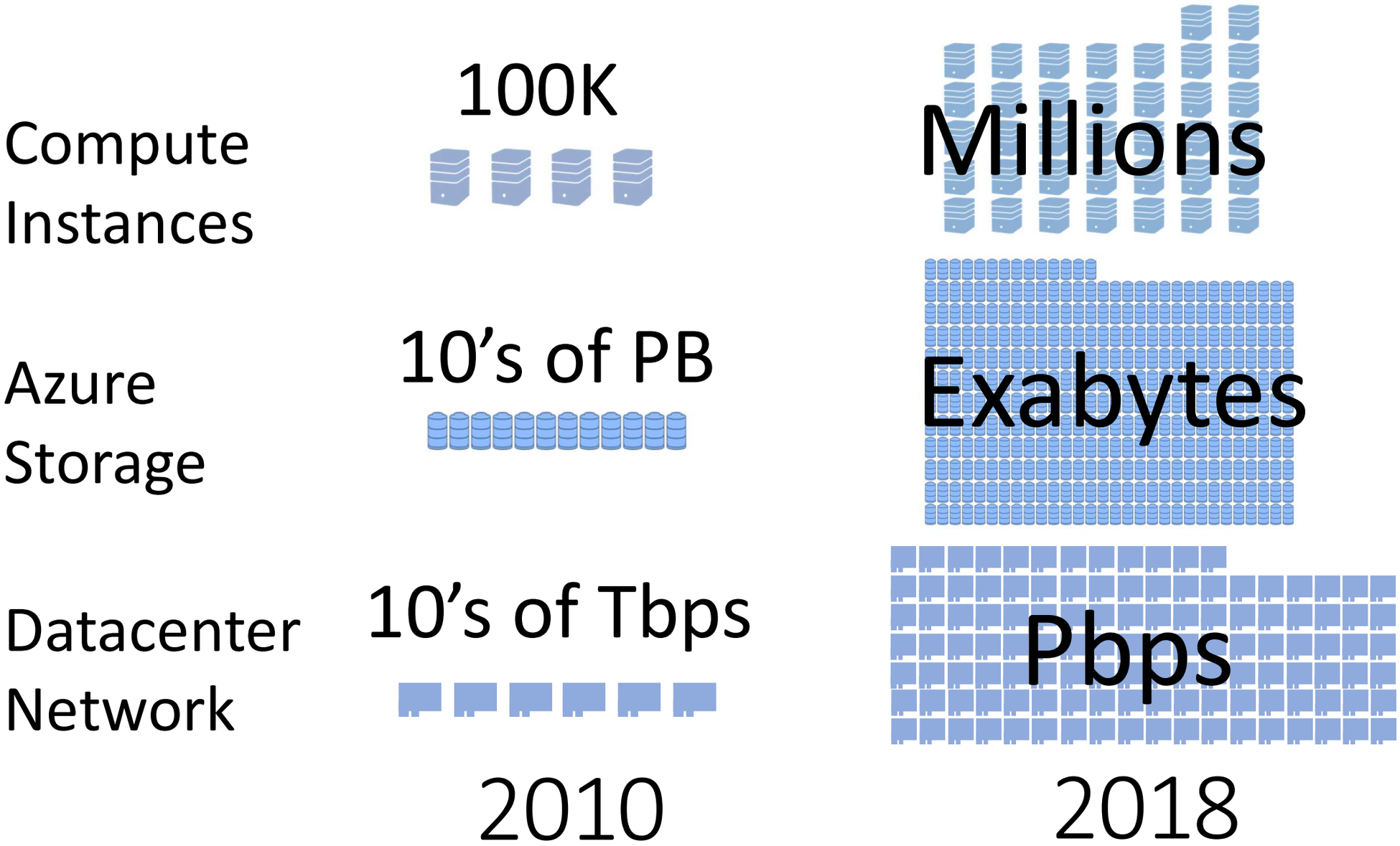


What is Scale?

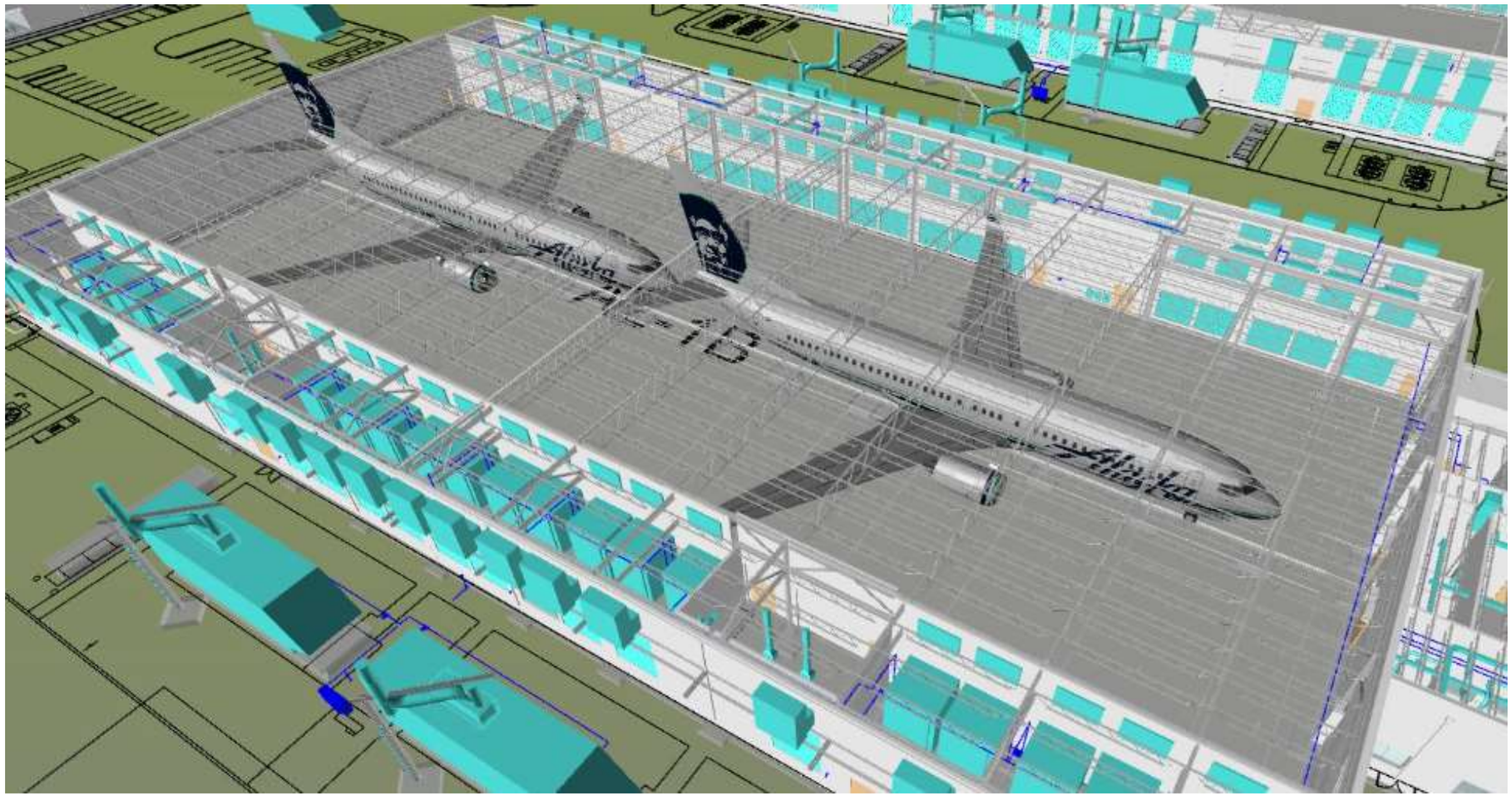


54 Global Azure Regions
100s of DCs

2013
2014
2015
2016
2017
Coming Soon


Compute Instances
100K
Millions
Azure Storage
10’s of PB
Exabytes
Datacenter Network
10’s of Tbps
Pbps
2010
2018

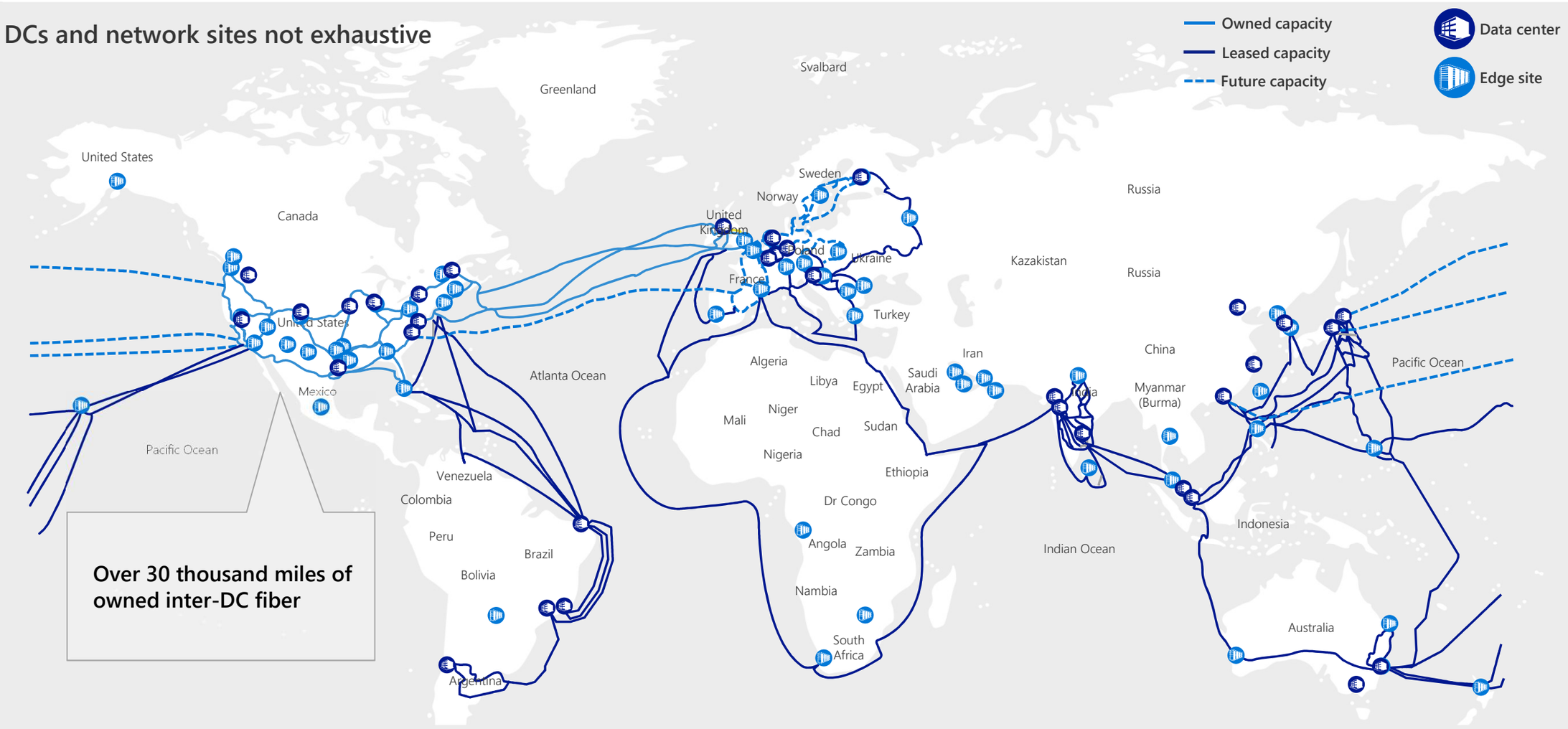

Quincy, WA




Cheyenne, WY


Cheyenne, WY

Azure inter-DC fiber backbone



Azure Scale & Momentum

>85%
Fortune 500 using Microsoft Cloud

>120,000
New Azure customers a month

>9 MILLION
Azure Active Directory Orgs

>18 BILLION
Azure Active Directory authentications/week

> 3 TRILLION
Azure Event Hubs events/week

>60 TRILLION
Azure storage objects

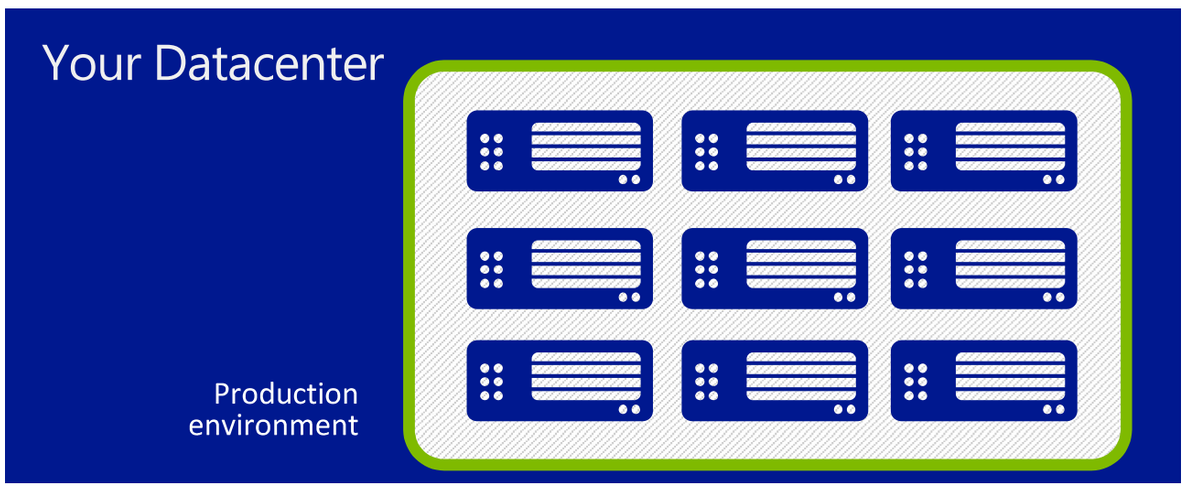
>900 TRILLION
requests/day

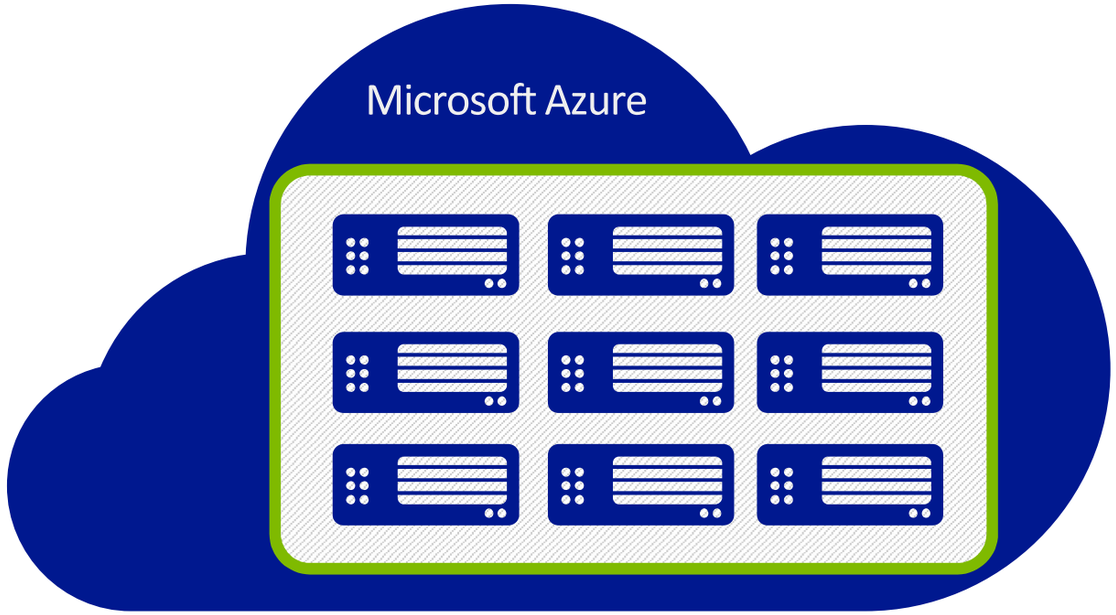
>50% of Azure VMs are Linux VMs

>110 BILLION
Azure DB requests/day

Infrastructure as a Service: Just buy some VMs

Deploy VMs anywhere
with no lock-in





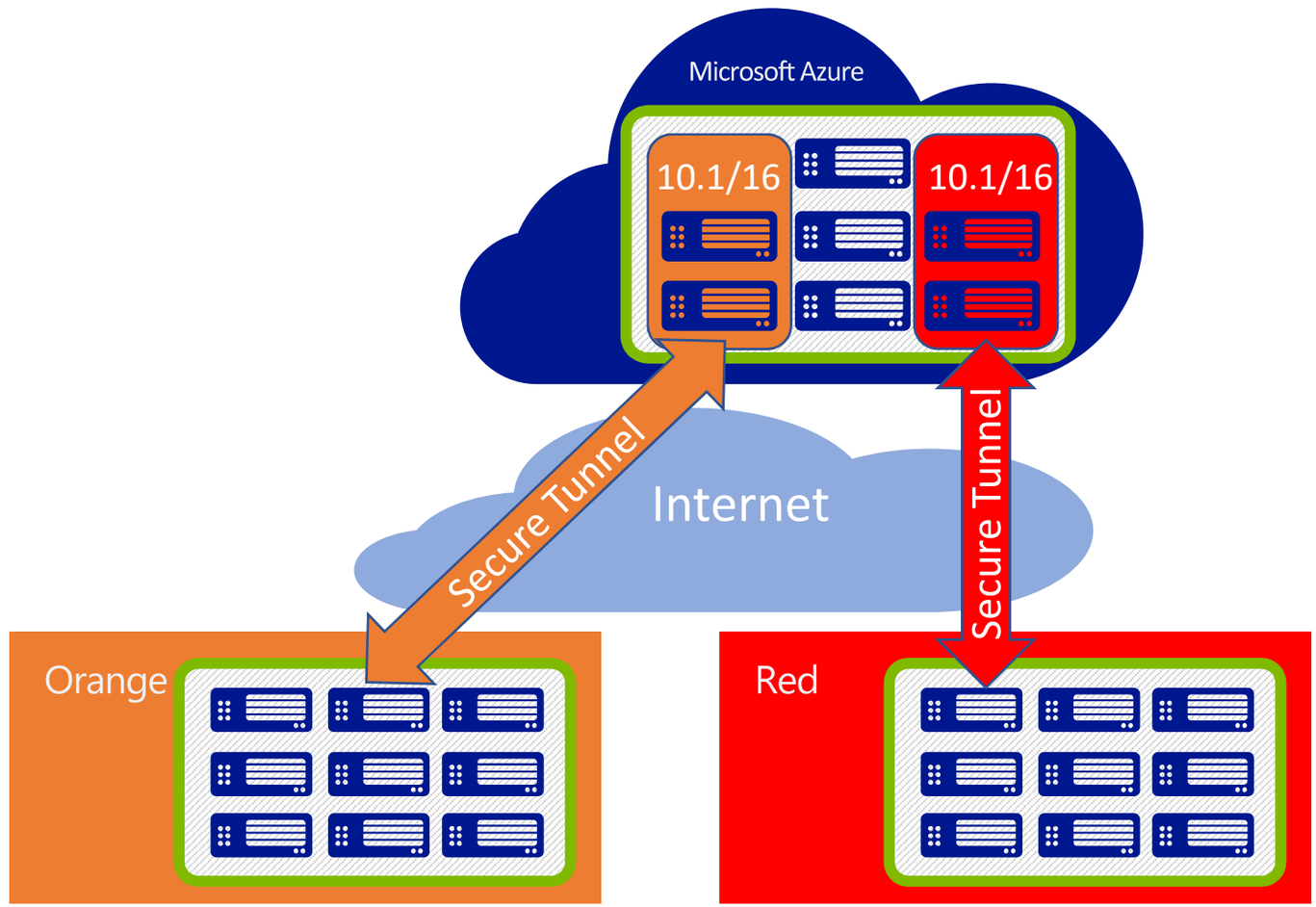
Easy VM portability

If it runs on Hyper-V/Xen/KVM, it runs in Azure:
Linux, Windows, …

What Does IaaS Mean for Networking? Scenario: BYO Network

Azure Virtual Networks

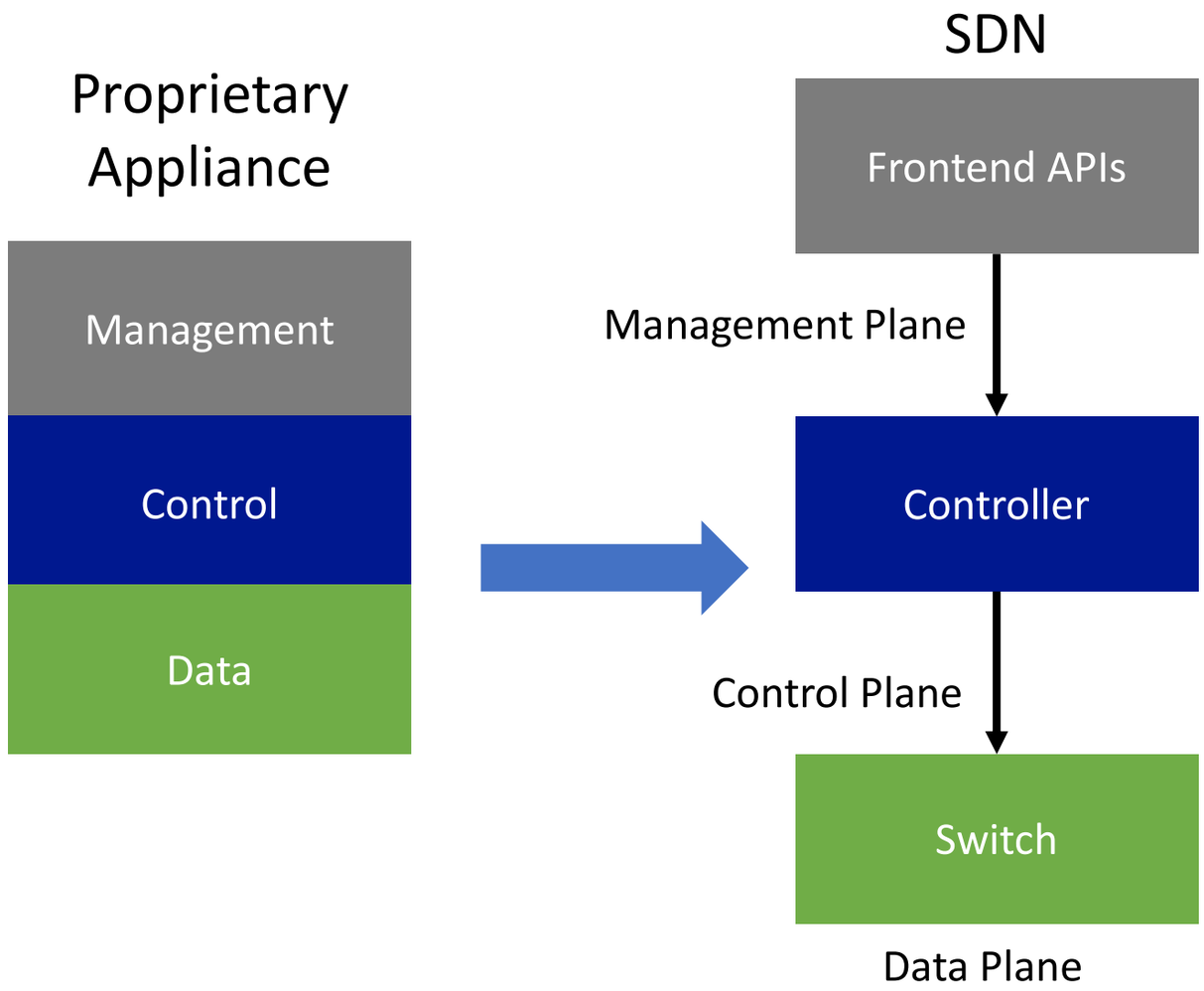
- Goal: BYO Address Space + Policy
- Azure is just another branch office of your enterprise, via VPN
- Communication between tenants of your Azure deployment should be efficient and scalable



How do we support O(1M) virtual networks, spread over a region of O(1M) server DCs?

Start by finding the right abstractions

Software Defined Networking: Building the Right Abstractions



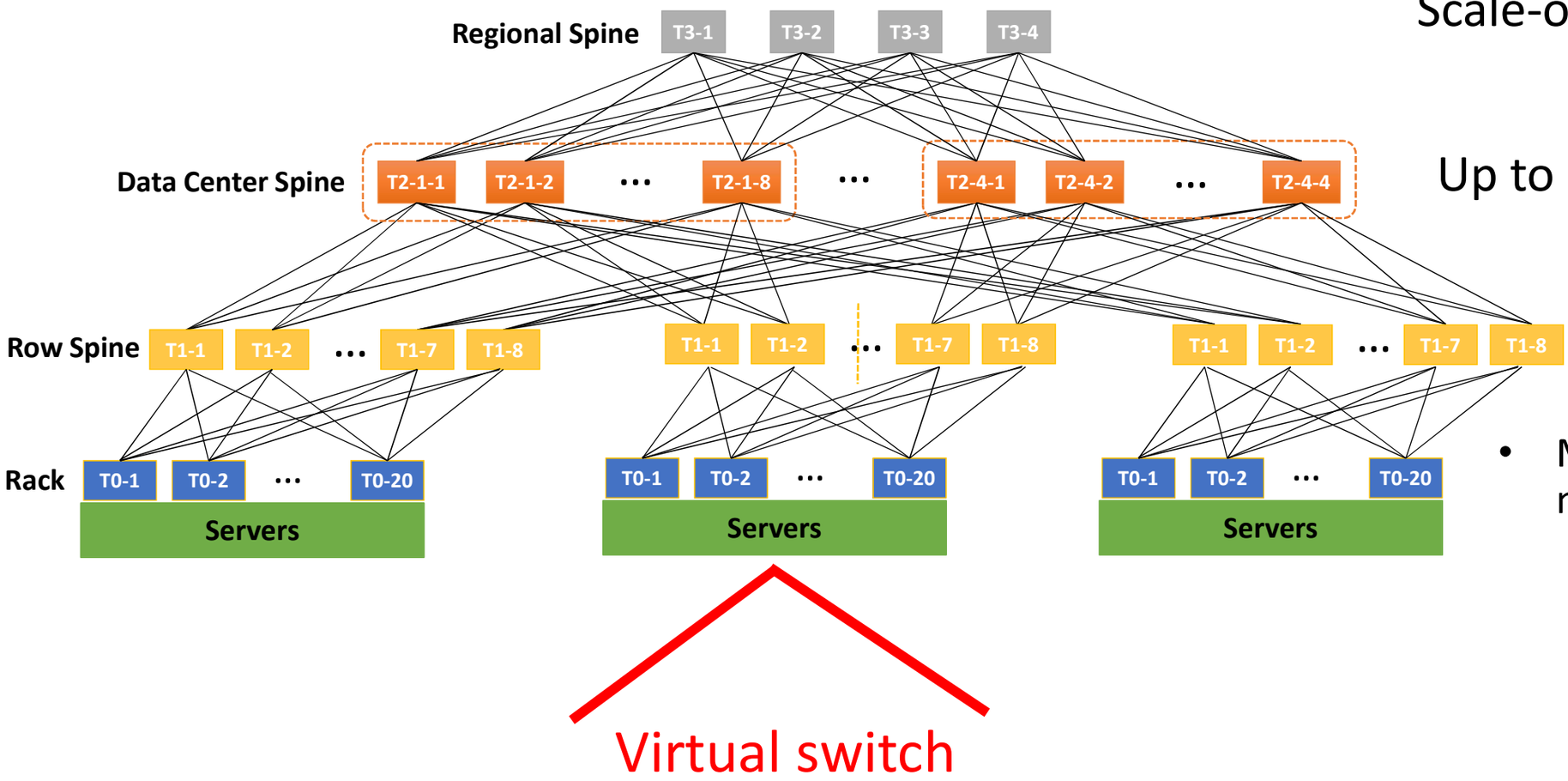
Example: ACLs

Management plane	Create a tenant
Control plane	Plumb tenant ACLs to switches
Data plane	Apply ACLs to flows

Data plane needs to apply per-flow policy to millions of VMs

How do we apply billions of flow policy actions to packets?

Azure Data Center Network Fabrics with 50G NICs



Scale-out, active-active

Up to 128 switches wide!

- Microsoft software on merchant silicon
  - Switch Abstraction Layer (SAI)
  - SONiC: Linux-based Switch firmware
  - OCP support

Broadcom Tomahawk III: 32x400GbE on a single chip



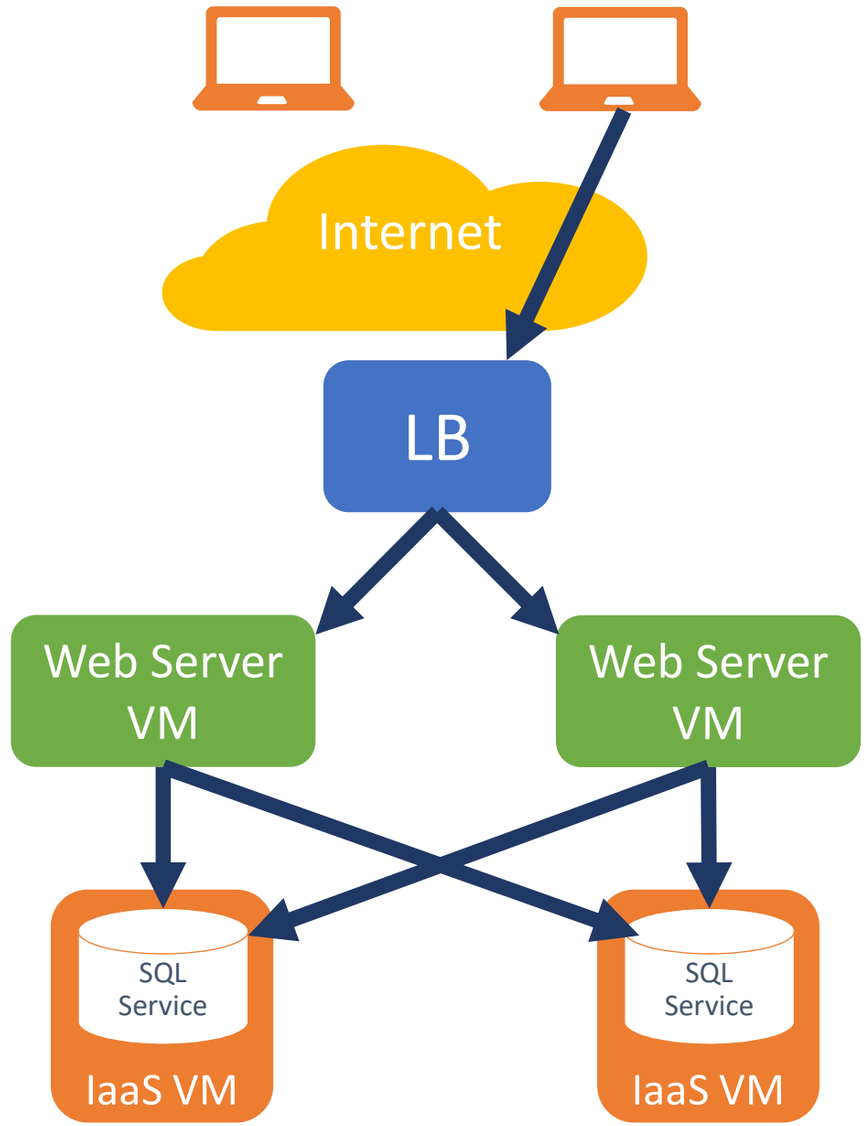
Amazing scale and performance – but can’t hold much policy per-VM on a chip at this scale

Why do we need Virtual Switch SDN Policy?

Policy application at the host is more scalable!

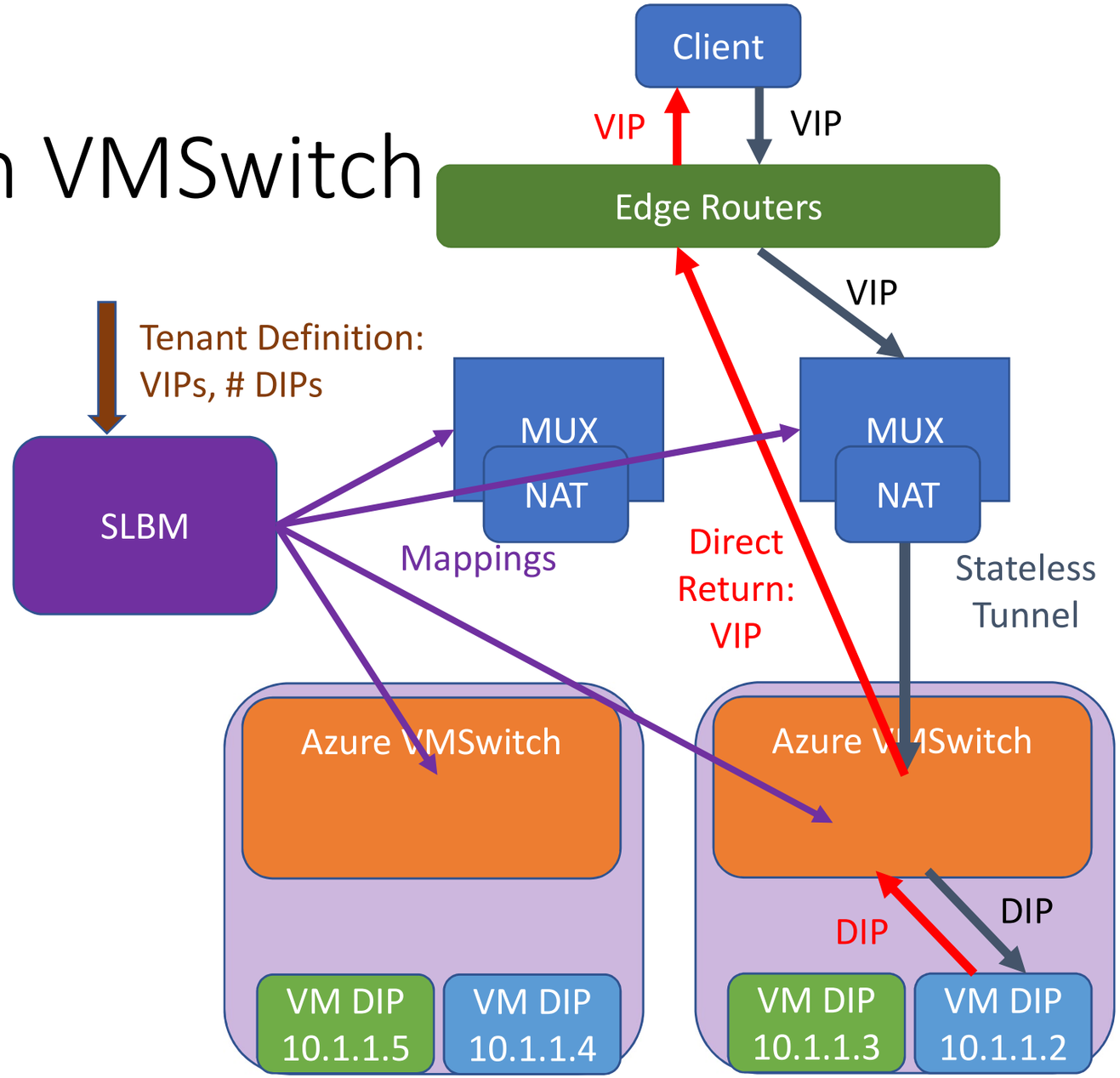
Example #1: LB (From Ananta, SIGCOMM '13)

- All infrastructure runs behind an LB to enable high availability and application scale
- How do we make application load balancing scale to the cloud?
- Challenges:
  - How do you load balance the load balancers?
  - Hardware LBs are expensive, and cannot support the rapid creation/deletion of LB endpoints required in the cloud
  - Support 10s of Gbps per cluster
  - Need a simple provisioning model



“SDN” Approach: Software LB with NAT in VMSwitch

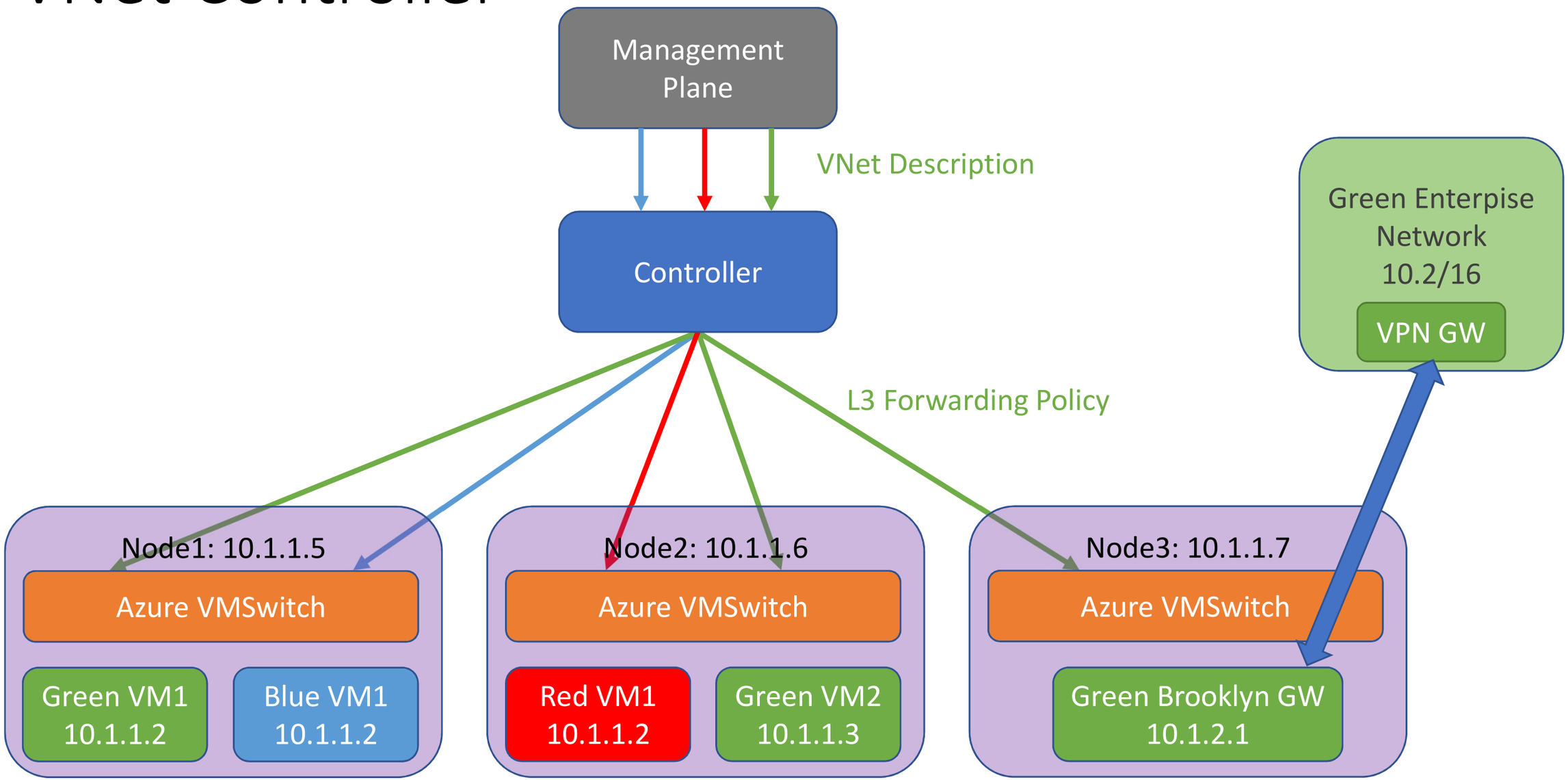
- Goal of an LB: Map a Virtual IP (VIP) to a Dynamic IP (DIP) set of a cloud service
- Two steps: Load Balance (select a DIP) and NAT (translate VIP->DIP and ports)
- Pushing the NAT to the vswitch makes the MUXes stateless (ECMP) and enables direct return
- Single controller abstracts out LB/vswitch interactions



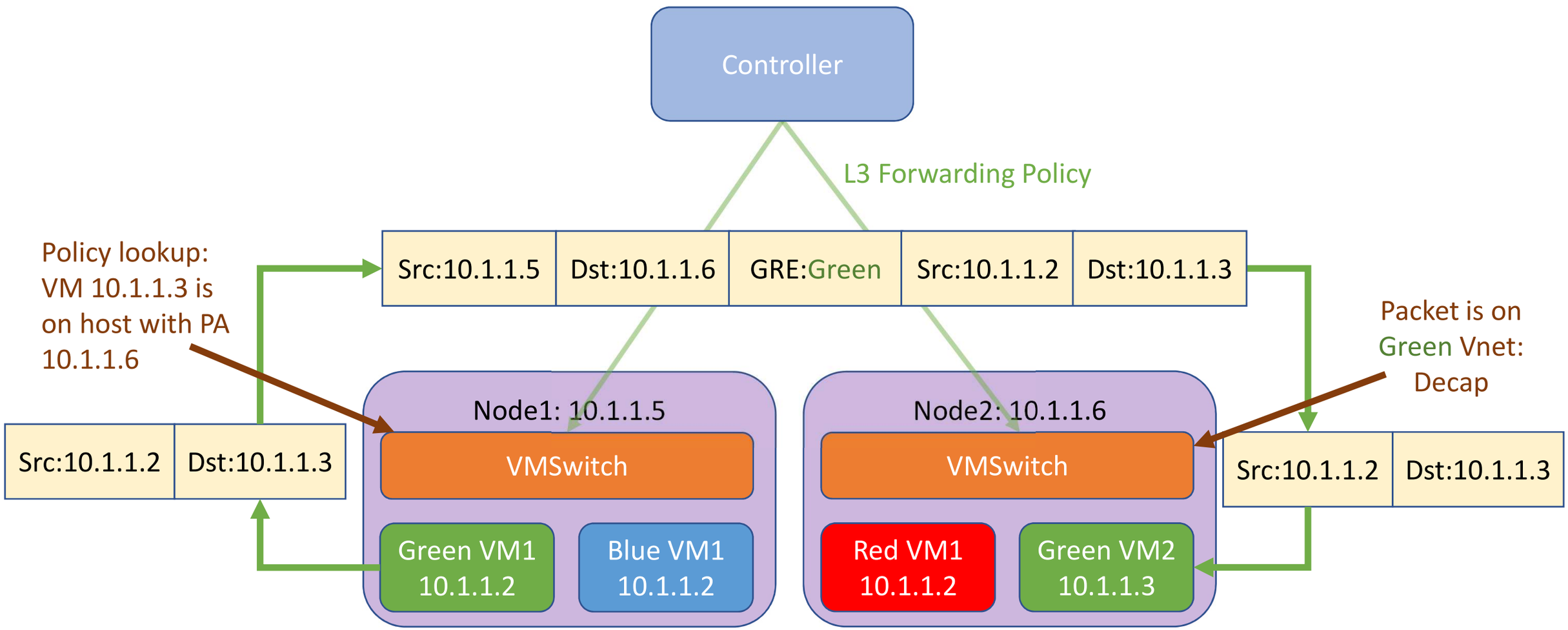
Example #2: Virtual Networking (Vnet)

- Ideas from VL2 (SIGCOMM ‘09)
- Goal is to map Customer Addresses (e.g. BYO IP space) to Provider Addresses (real 10/8 addresses on the physical network)
- This requires a translation of *every* packet on the network – no hardware device on our network is scalable enough to handle this load along with all of the relevant policy
- Enables companies to create their own virtual network in the cloud, defining their own topologies, security groups, middleboxes and more

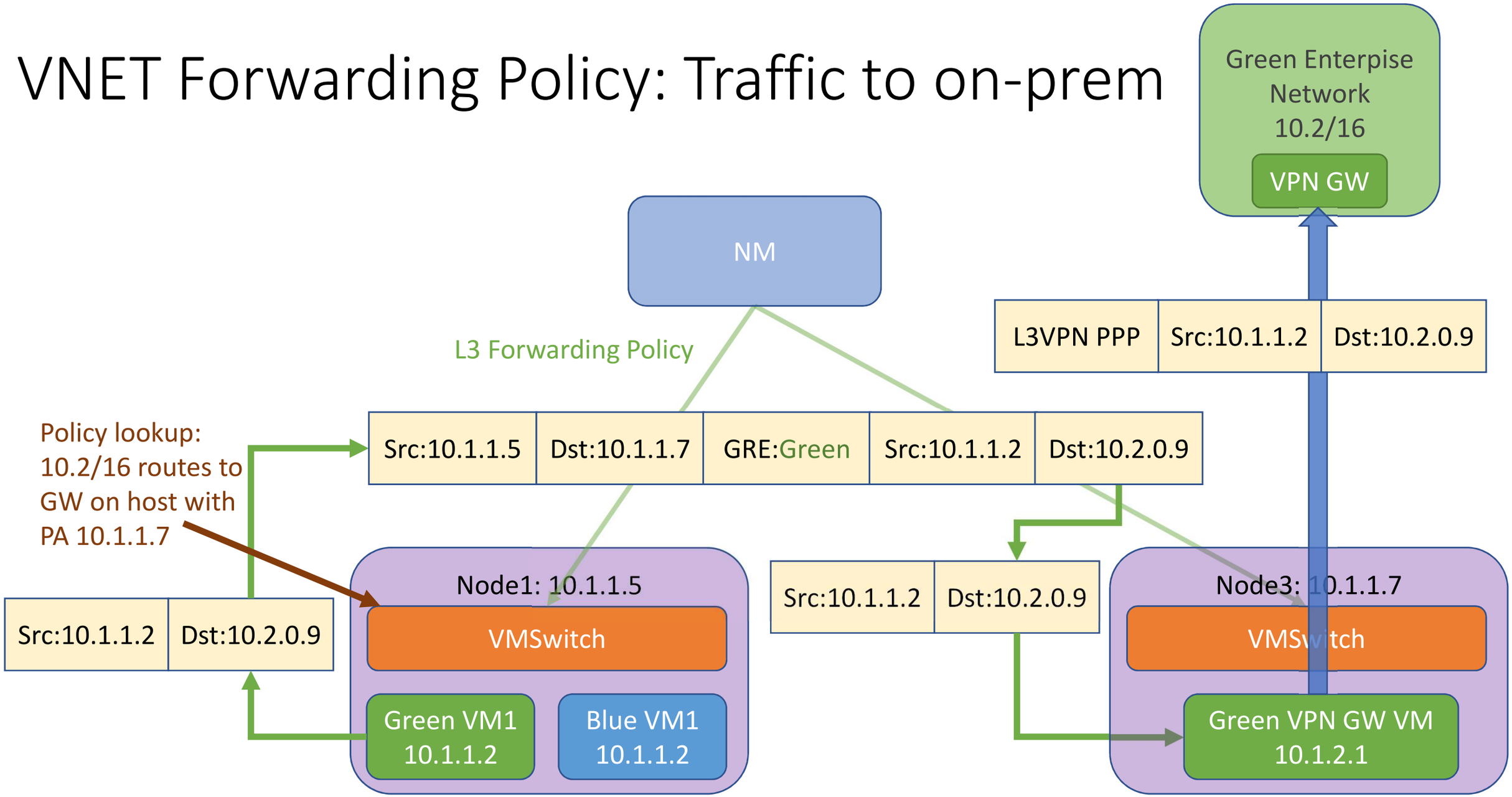
VNet Controller



Forwarding Policy: Intra-VNet

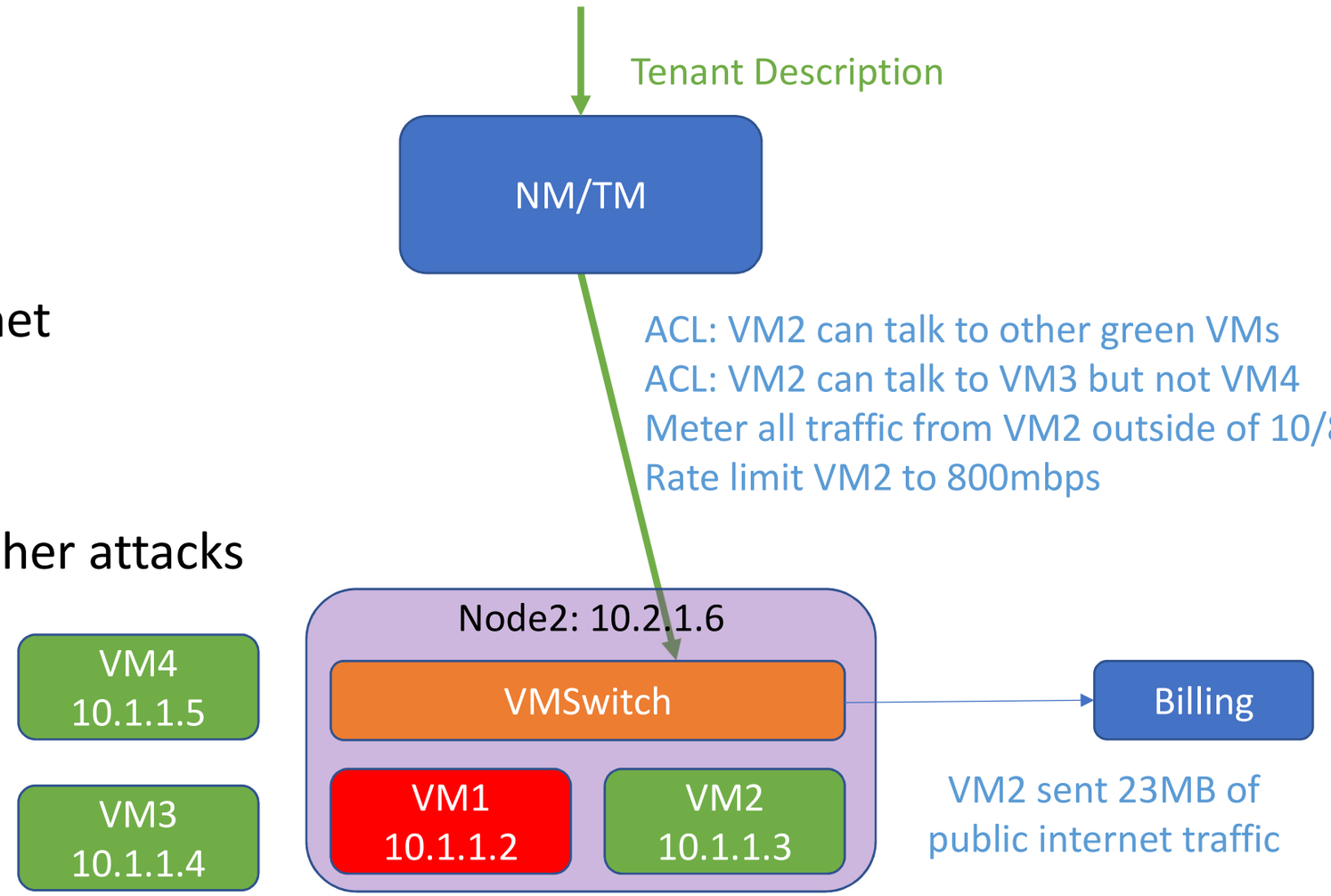


VNET Forwarding Policy: Traffic to on-prem



Even More VSwitch…

- 5-tuple ACLs
  - Infrastructure Protection
  - User-defined Protection
- Billing
  - Metering traffic to internet
- Rate limiting
- Security Guards
  - Spoof, ARP, DHCP, and other attacks
- More in development all the time…



Proliferation of Virtual Switches

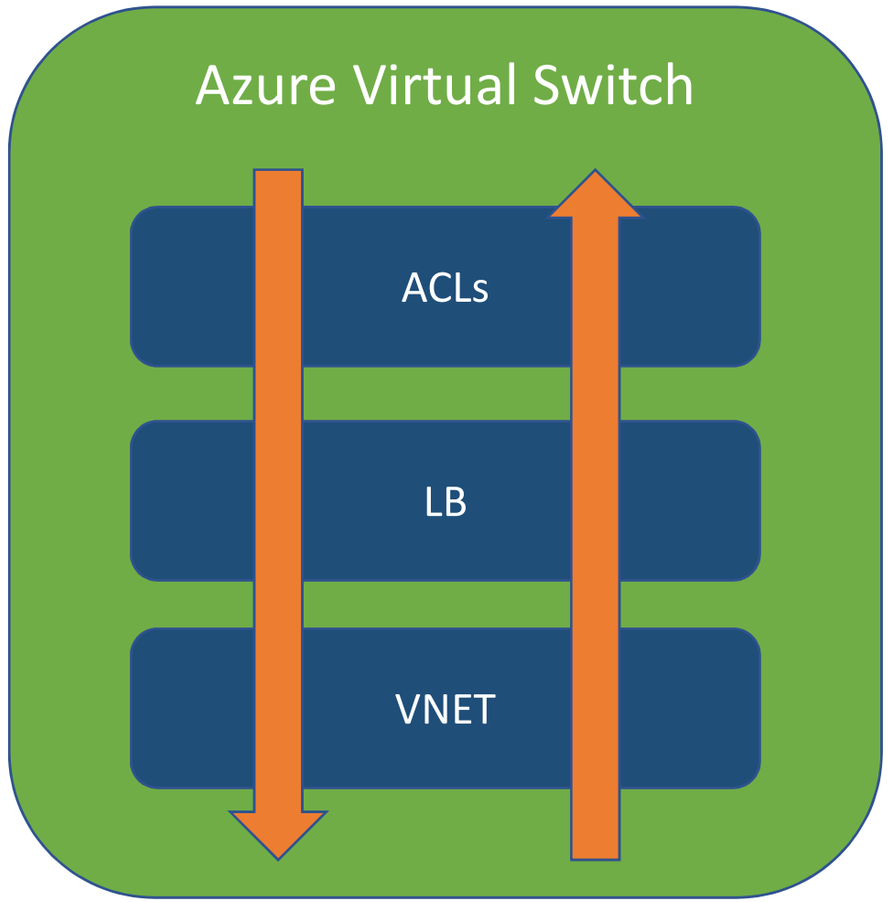
- Open vSwitch – popular open source vswitch (used in e.g. OpenStack)
- Based on OpenFlow – a popular early SDN protocol for programming switches with flow actions
- Microsoft Azure VFP, Amazon Annapurna NIC, Google Andromeda – major clouds have built their own programmable host networks



Azure’s Programmable vSwitch

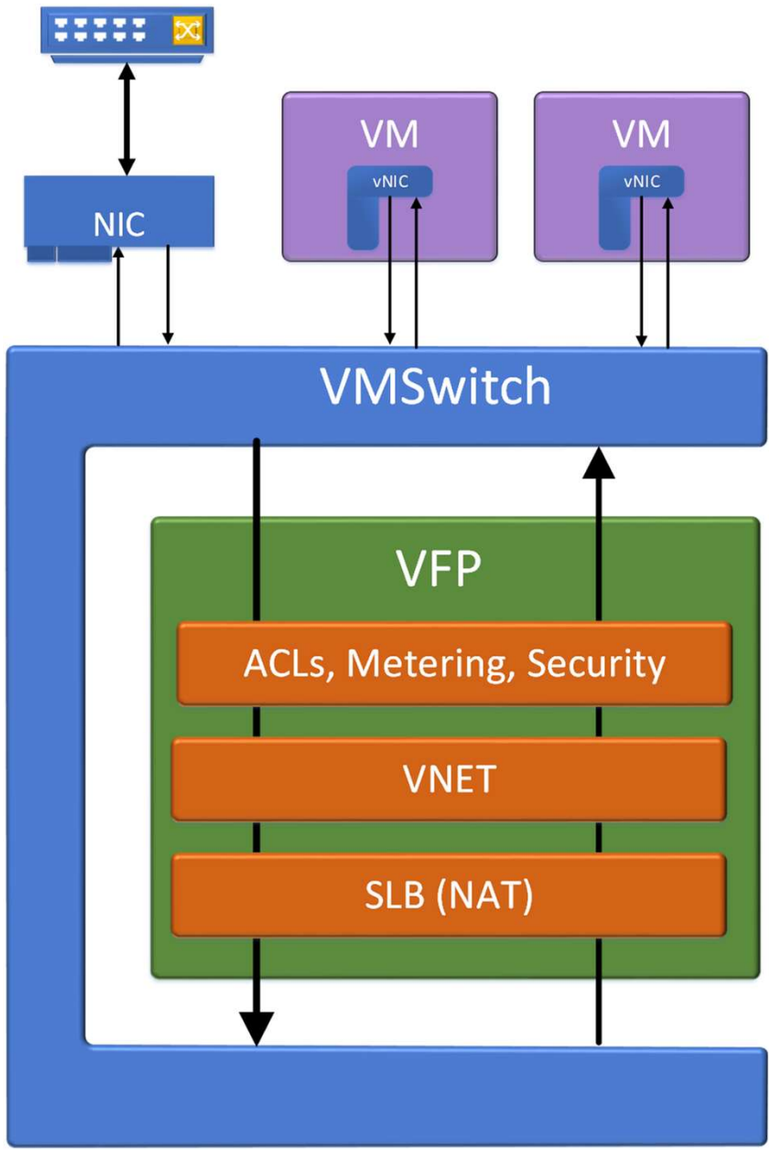
Early Approach to Azure Vswitch (2009-2011): Stacked SDN drivers per app

- Each SDN application is a driver module hard compiled into the vswitch, handling packets on its own
- Changes to SDN policy require kernel space changes, and an OS update
- Was revolutionary for us in shipping LB and VNET and Host SDN – but not easy to add new SDN Apps
- After a couple of years we decided we needed a more flexible Host SDN platform



Original Goals for Azure Host SDN Platform

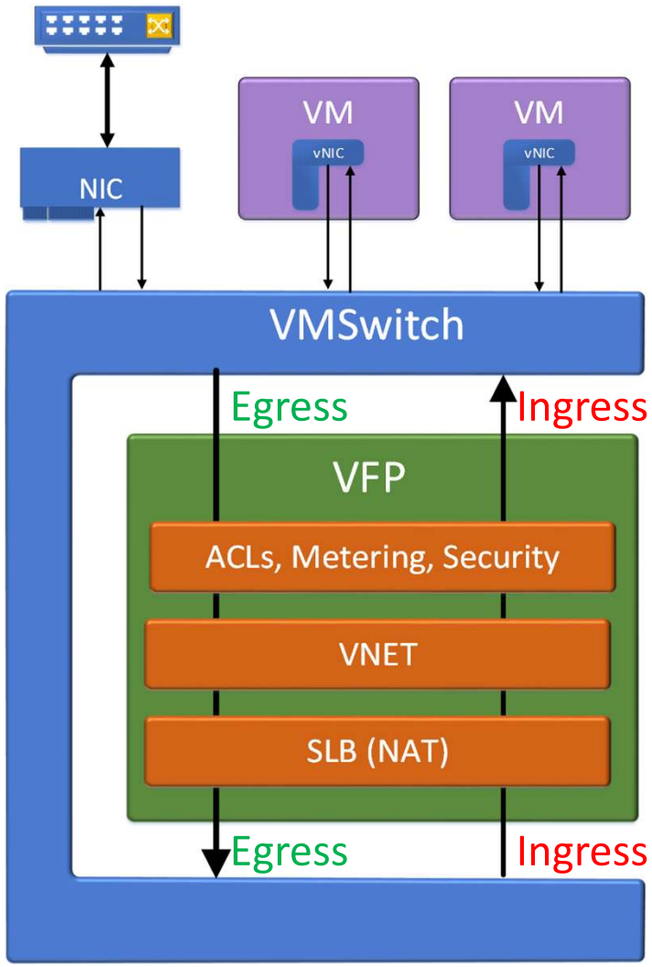
- **Goal 1:** Provide a programming model allowing for multiple simultaneous, independent network controllers to program network applications, minimizing cross-controller dependencies
- **Goal 2:** Provide a MAT programming model capable of using connections as a base primitive, rather than just packets – stateful rules as first class objects
- **Goal 3:** Provide a programming model that allows controllers to define their own policy and actions, rather than implementing fixed sets of network policies for predefined scenarios



Virtual Filtering Platform (VFP) Azure's SDN Dataplane

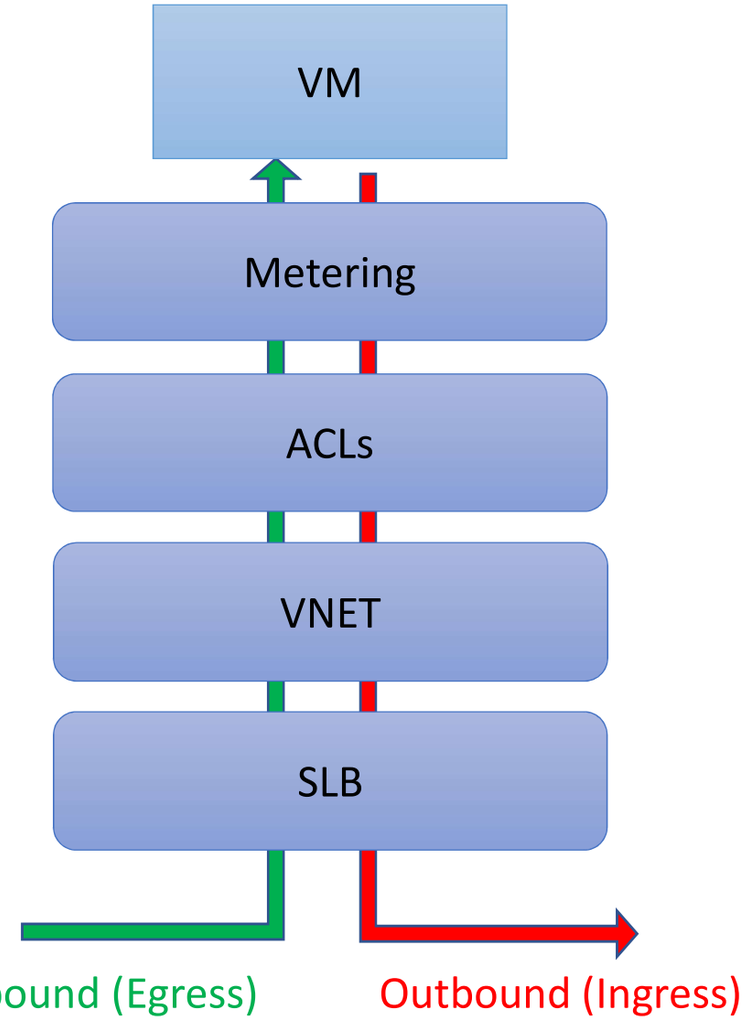
- Plugin module for WS2012+ VMSwitch
- Provides core SDN functionality for Azure networking services, including:
  - Address Virtualization for VNET
  - VIP -> DIP Translation for SLB
  - ACLs, Metering, and Security Guards
- Uses programmable rule/flow tables to perform per-packet actions
- Programmed by multiple Azure SDN controllers, supports all dataplane policy at line rate with offloads

VFP Translates L2 extensibility (ingress/egress to switch) to L3 extensibility (inbound/outbound to VM)



Egress -> Inbound

Ingress -> Outbound

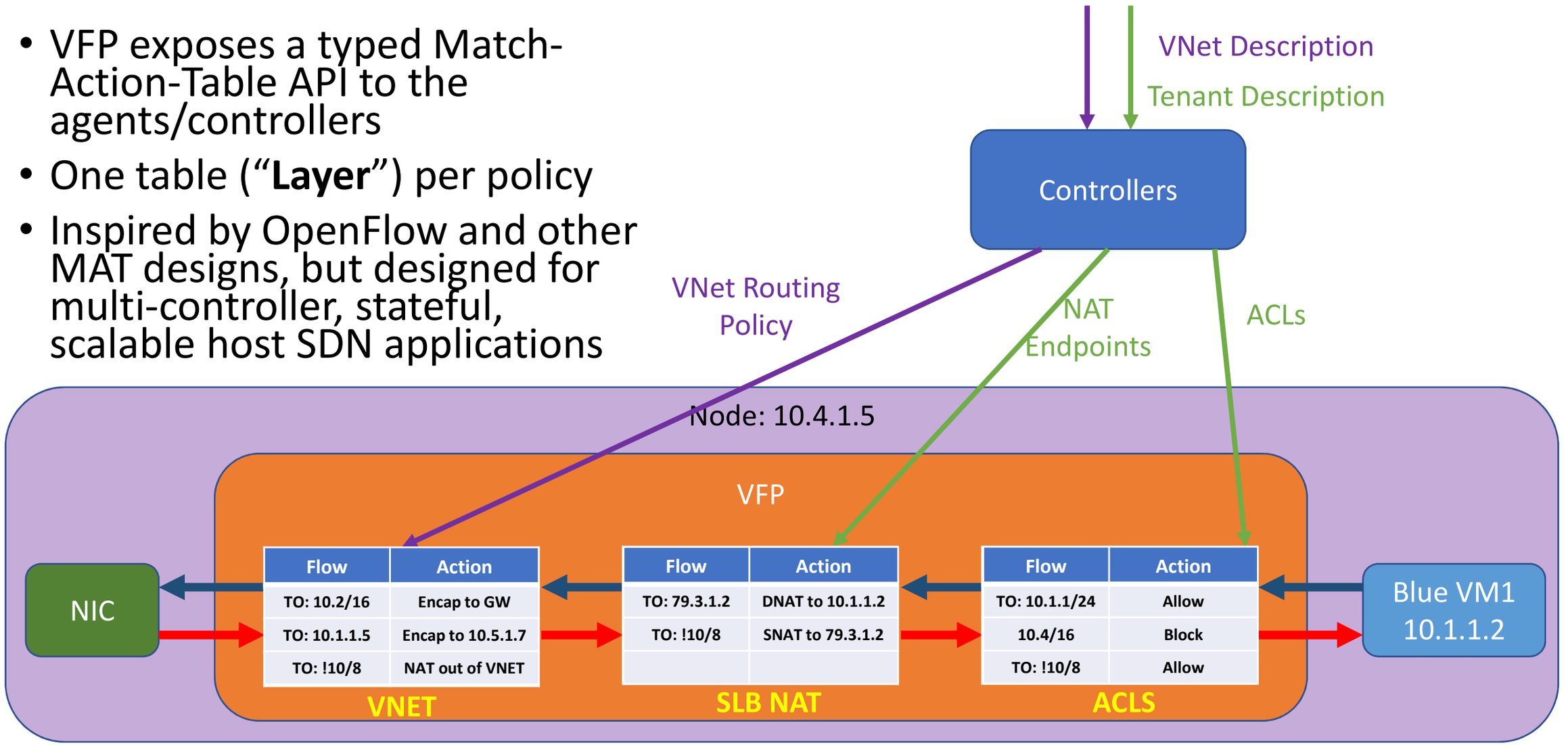


Goal: All Policy is in the Controller - VFP is a Fast, Flexible Implementation of Policy

- To enable agility, allow controllers to specify exactly what they want to do at the flow/packet level, so they can implement new SDN scenarios without dataplane driver changes
- VFP focuses on integrating multi-controller policies and scaling the host dataplane – perf and offloads without sacrificing flexibility
- 3 Key Primitives we expose to controllers:
  - **Layers** – independent flow tables per controller to order the pipeline
  - **Rule Matches** – define which packets match which rule
  - **Rule Actions** – what to do with a packet for a given rule

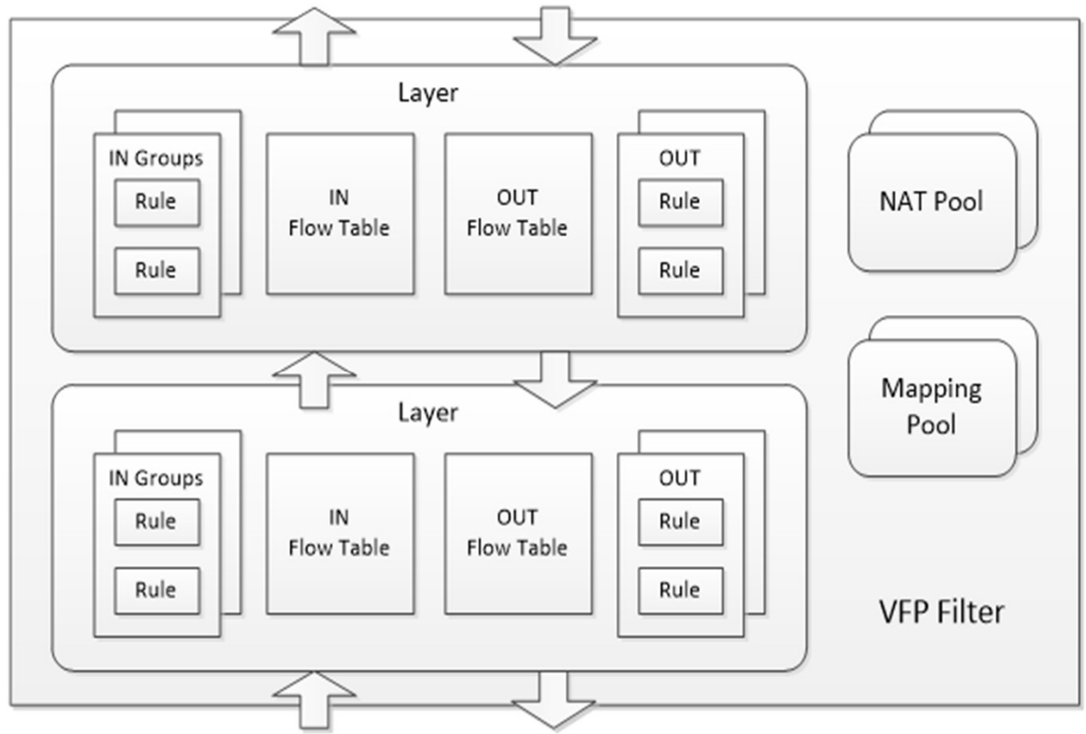
Key Primitive: Match Action Tables

- VFP exposes a typed Match-Action-Table API to the agents/controllers
- One table ("**Layer**") per policy
- Inspired by OpenFlow and other MAT designs, but designed for multi-controller, stateful, scalable host SDN applications



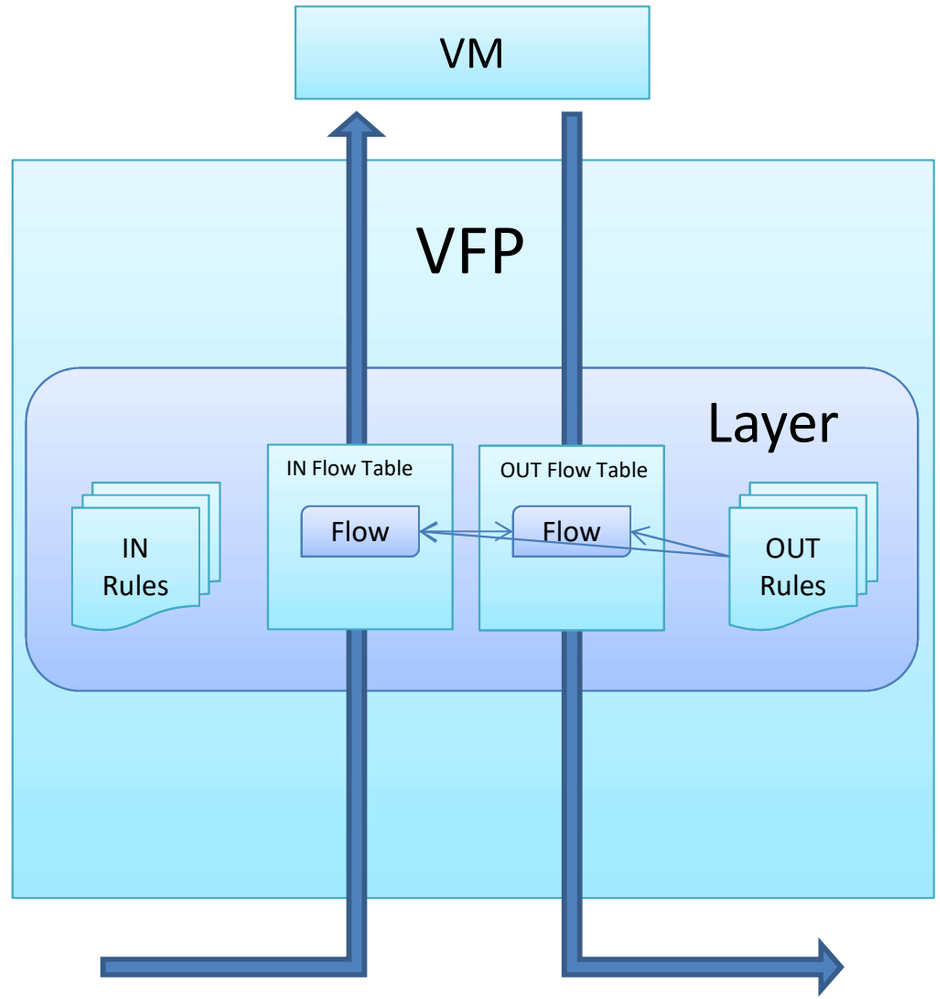
Layers

- A VFP layer is not a built-in function – it is a generic set of rule/flow tables
- Any layer can be created at any time – it is only an “LB layer” or a “VNET layer” based on what rules are plumbed into it
- Resources like NAT pools or PA->CA mapping pools are available to any layer to implement special functionality (e.g. SLB or VNET)

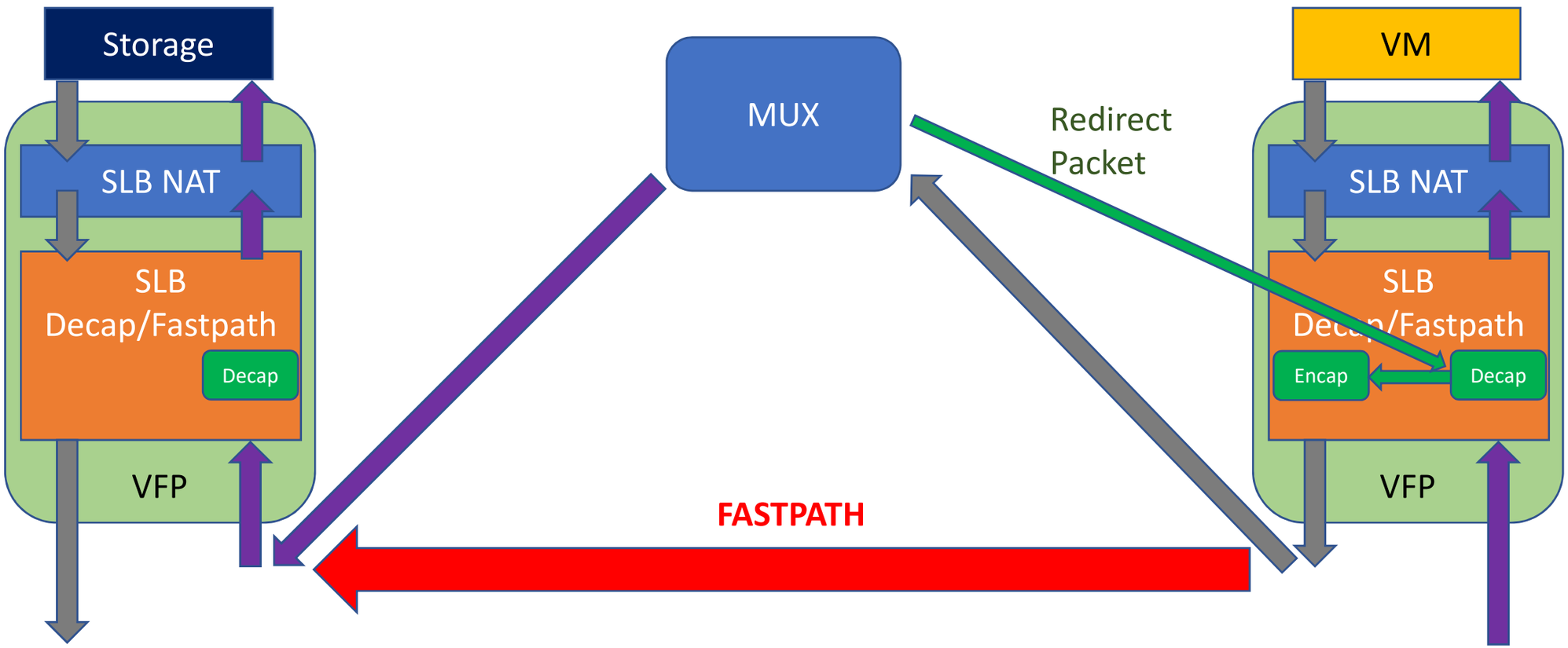


Everything is Stateful

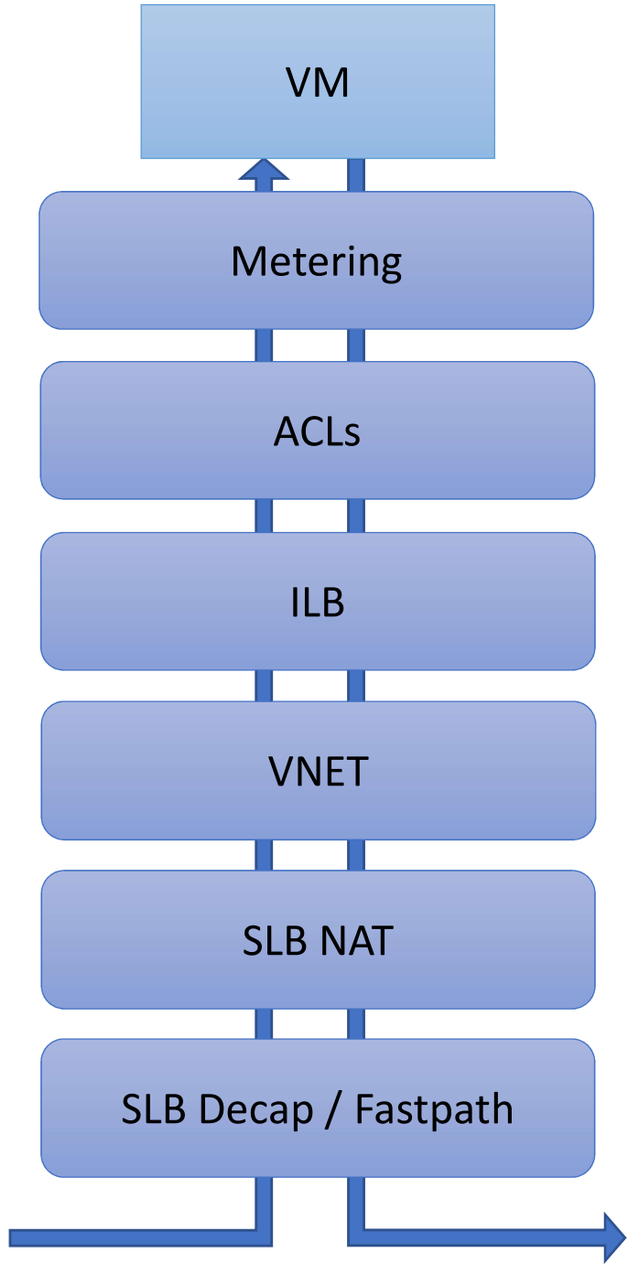
- The core primitive of most policy is a (TCP, UDP, …) connection – translates to a two-way flow
- 5-tuple ACLs, VIP-DIP SLB NAT, dynamic outbound SNAT, and more
- Stateful rules make it easy to reason about asymmetric policy – rules apply to whichever side started the flow, and the reverse happens automatically for the other direction
- Flow state managed by TCP connection tracker



Cool Uses of Stateful Flows – LB Fastpath



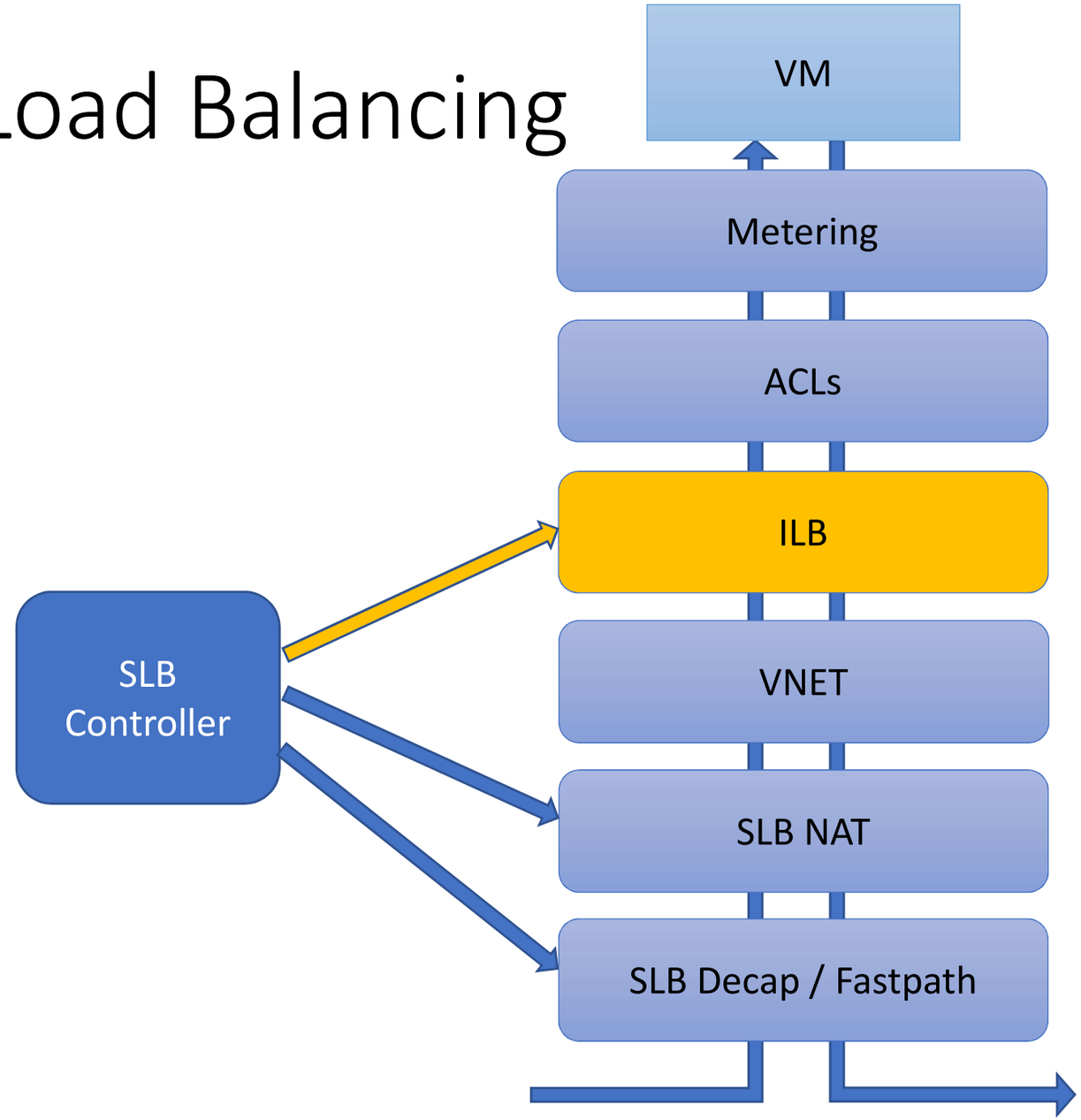
Example VFP Layers:
Support for LB, VNET, Security Groups, and Billing



Successfully deployed across Azure in 2012

Agility Example: Internal Load Balancing

- LB team wanted to offer CA-space LB in addition to PA-space LB
- All they had to do was create a new layer – added new policy by specifying CA-space rule matches for NAT rules
- **No new work in VFP, because we picked the right primitives**



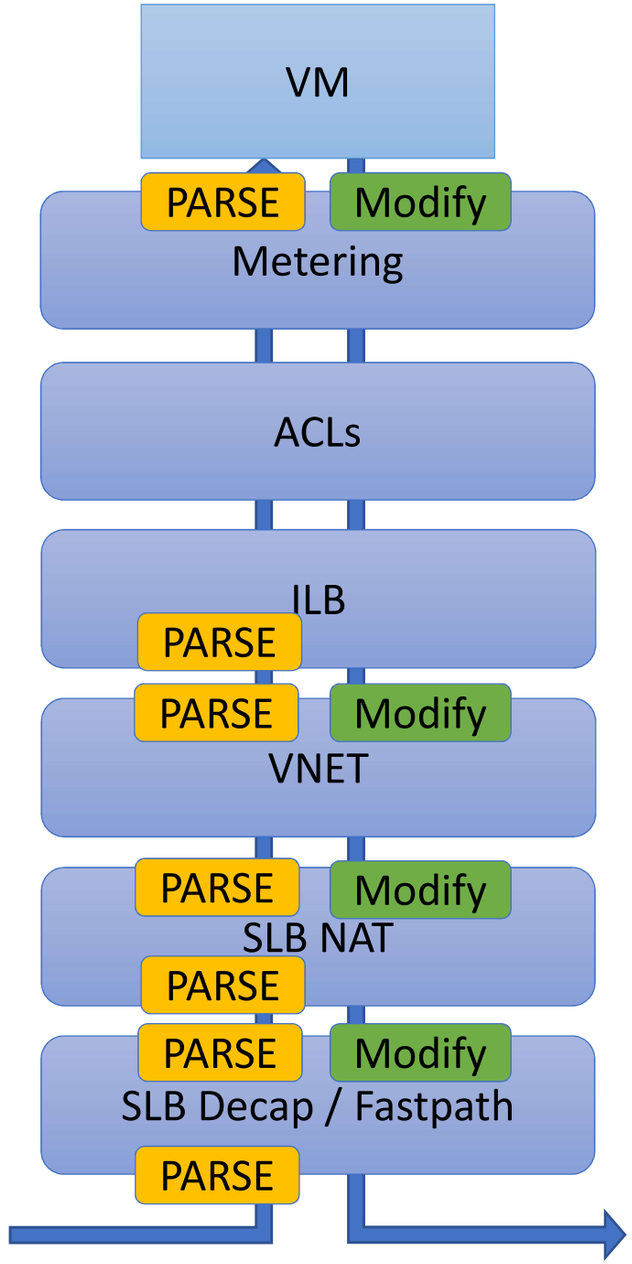
New Goals for VFPv2 (2013-2014)

- **Goal 4:** Provide a serviceability model allowing for frequent deployments and updates without requiring reboots or interrupting VM connectivity for stateful flows, and strong service monitoring
- **Goal 5:** Provide very high packet rates, even with a large number of tables and rules, via extensive caching
- **Goal 6:** Implement an efficient mechanism to offload flow policy to programmable NICs, without assuming complex rule processing

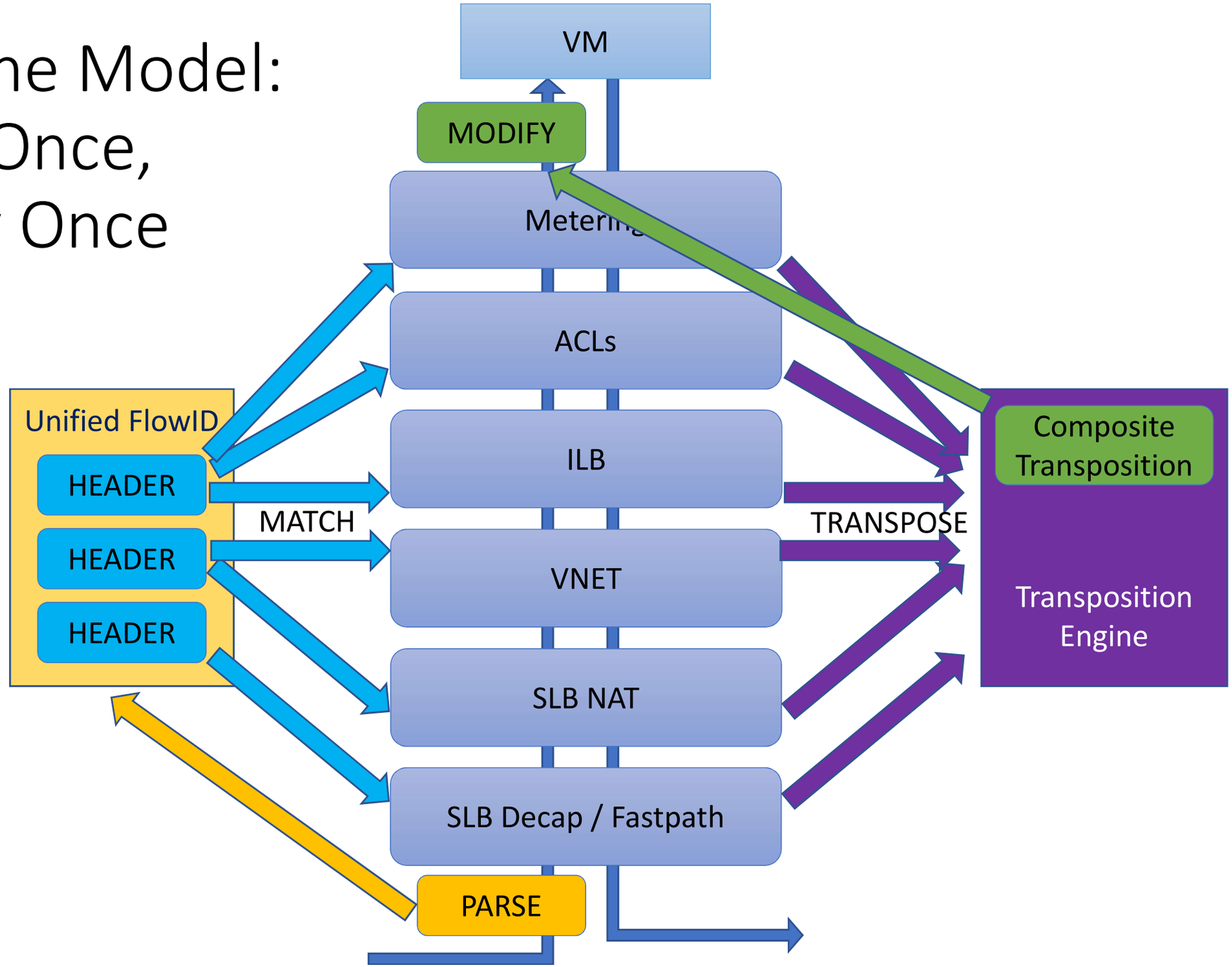
VFPv1 Layers - Challenges

- Holdover from original vswitch design – every layer independently handles, parses, and modifies packets
- Most of our layers want to be stateful – but this means independent connection tracking and flow state at each layer
- As host SDN became easy to program and widely used, people wanted to add new layers all the time
- Couldn’t keep adding layers and scaling up

We need a better primitive for actions!



ASIC Pipeline Model: Parse Once, Modify Once

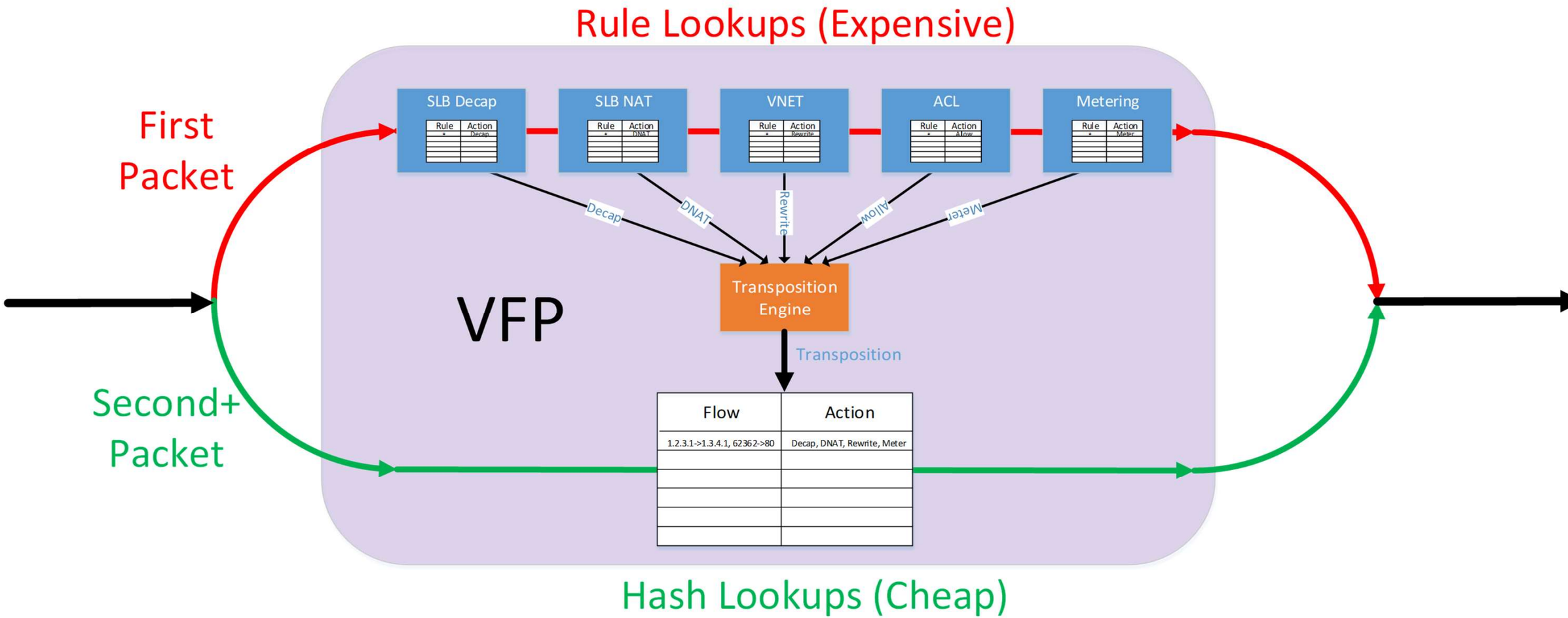


Shipped in 2014

Header Transpositions Complete our Generic Programmable Network API Story

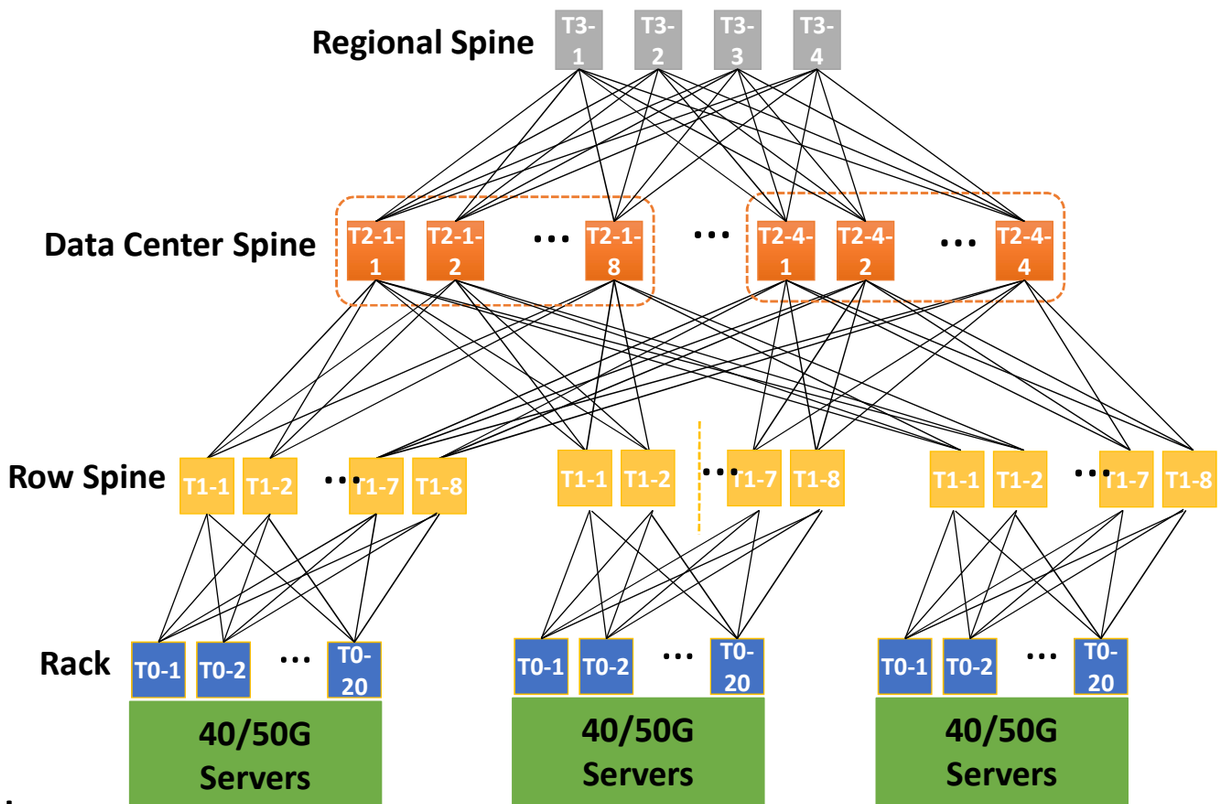
- In order to enable agility, we want controllers to be able to define new types of policy dynamically without needing to change VFP.
- We already provide flexibility in:
  - **Layers**: Controllers can define new layers dynamically for their own policy without interfering with other controllers' layers
  - **Rules:** Controllers can define which rules match which packets via a consistent 5-tuple match API, nothing specific to special policies
- **Header transpositions** provide the key third primitive: Ability to specify what exactly a rule does once it is matched
- All built in rules define HTs, but controllers can define their own rules by creating new ones out of HTs on the fly

Unified Flow Tables – A Fastpath Through VFP



Host Networking made our Physical Network Fast and Scalable

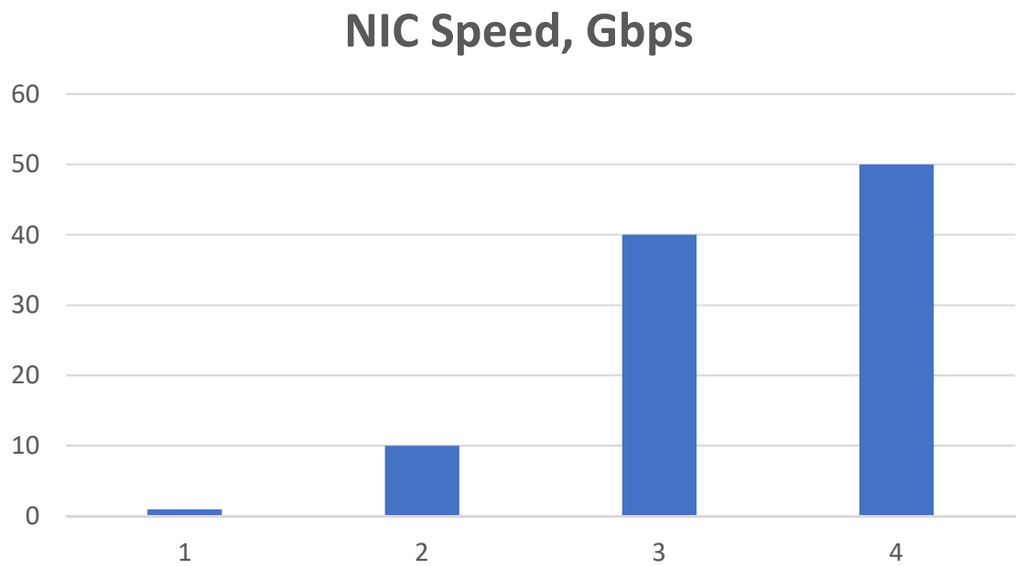
- Massive, distributed 40/100GbE network built on commodity hardware
  - No Hardware per tenant ACLs
  - No Hardware NAT
  - No Hardware VPN / overlay
  - No Vendor-specific control, management or data plane
- All policy is in software on hosts – and everything's a VM!
- Network services deployed like all other services
- VFP and network controllers, battle tested in the cloud, are available in Microsoft Azure Stack for private cloud as well



More scale pressure on the host…

Scaling Up SDN: NIC Speeds in Azure

- 2009: 1Gbps
- 2012: 10Gbps
- 2015: 40Gbps
- 2017: 50Gbps
- Soon: 100Gbps?

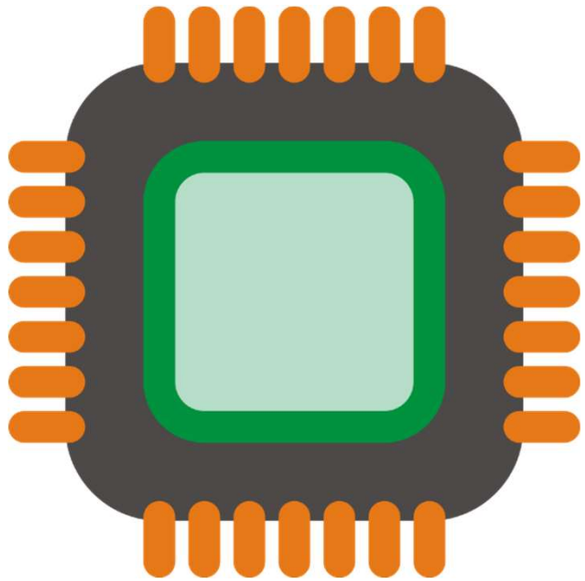


We got a 50x improvement in network throughput, but not a 50x improvement in CPU power!

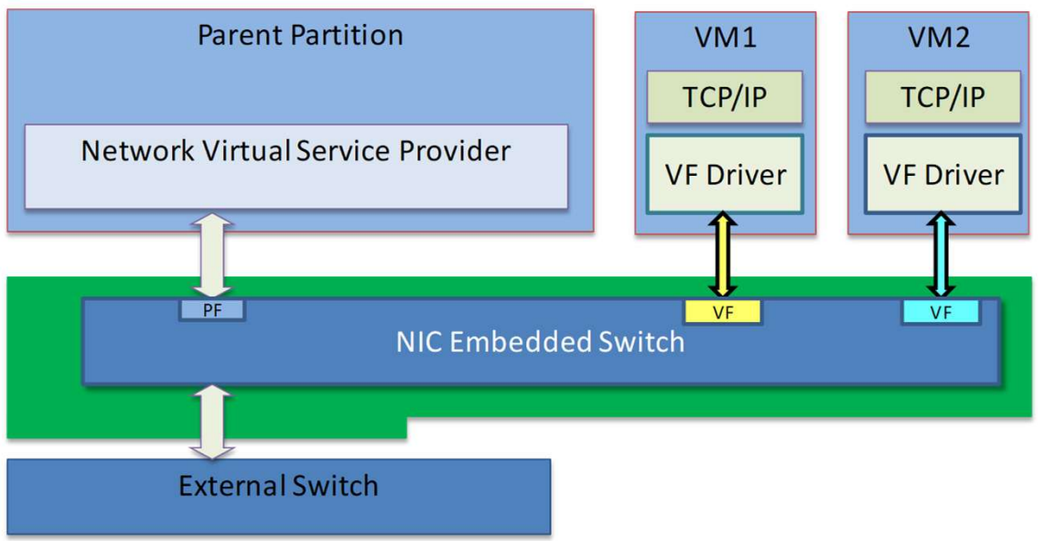
Host SDN worked well at 1GbE, ok at 10GbE… what about 40GbE+?

Traditional Approach to Scale: ASICs

- We've worked with network ASIC vendors over the years to accelerate many functions, including:
  - TCP offloads: Segmentation, checksum, …
  - Steering: VMQ, RSS, …
  - Encapsulation: NVGRE, VXLAN, …
  - Direct NIC Access: DPDK, PacketDirect, …
  - RDMA
- Is this a long term solution?



Example ASIC Solution:
Single Root IO Virtualization (SR-IOV) gives native performance for virtualized workloads



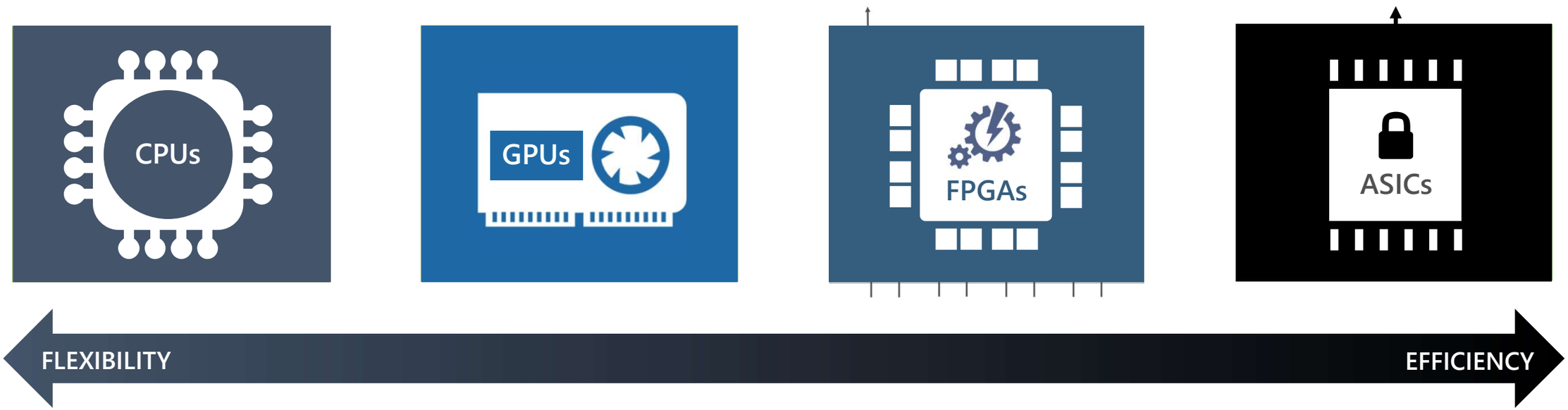
But where is the SDN Policy?

Hardware or Bust

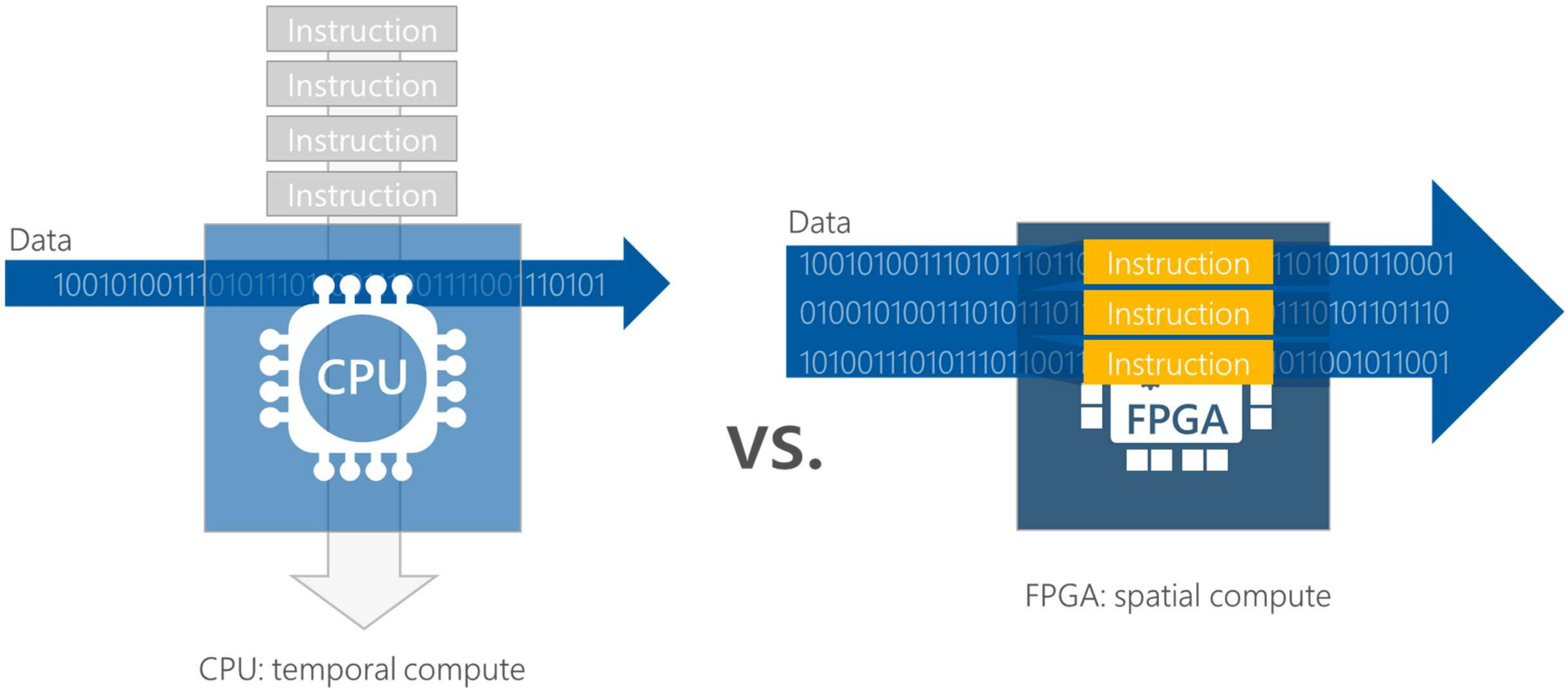
- SR-IOV is a classic example of an “all or nothing” offload – its latency, jitter, CPU, performance benefits come from skipping the host entirely
- If even one widely-used action isn’t supported in hardware, have to fall back to software path and most of the benefit is lost even if hardware can do 99% of the work
- Other examples: RDMA, DPDK, … a common pattern
- This means we need to consider carefully how we will add new functionality to our hardware as needed over time

How do we get the performance of hardware
with programmability of software?

Silicon alternatives

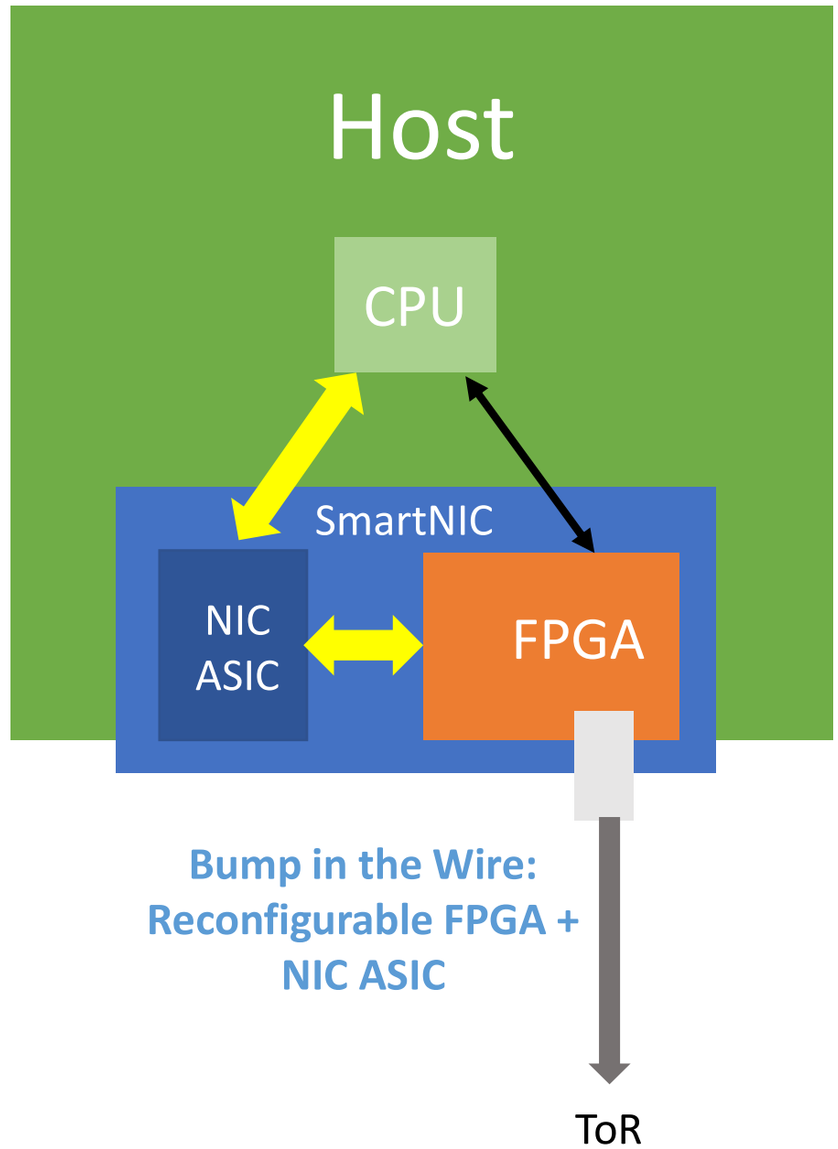


CPU vs. FPGA


Instruction
Instruction
Instruction
Instruction
Data
CPU
CPU: temporal compute
VS.
Data
Instruction
Instruction
Instruction
FPGA
FPGA: spatial compute

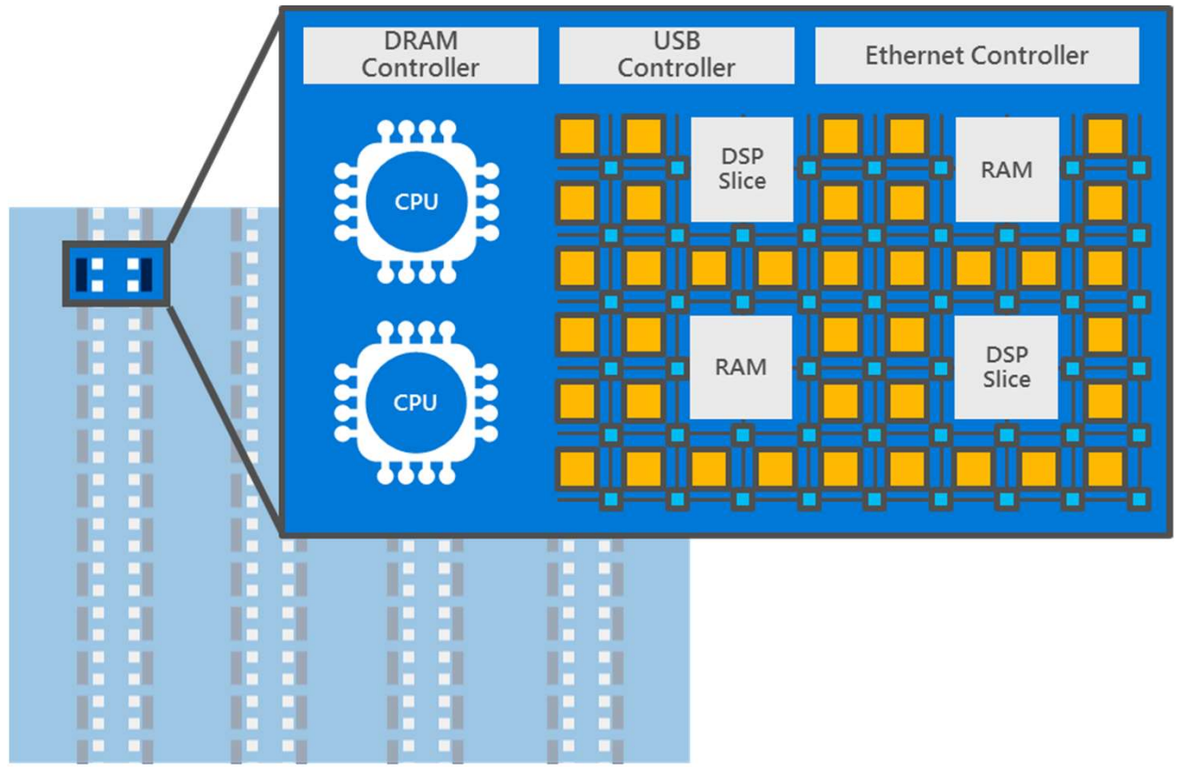
Our Solution:
Azure SmartNIC (FPGA)

- HW is needed for scale, perf, and COGS at 40G+
- 12-18 month ASIC cycle + time to roll new HW is too slow
- To compete and react to new needs, we need agility – SDN
- Programmed using Generic Flow Tables
  - Language for programming SDN to hardware
  - Uses connections and structured actions as primitives



What is an FPGA, Really?

- Field Programmable Gate Array
- Chip has large quantities of programmable gates – highly parallel
- Program specialized circuits that communicate directly
- Two kinds of parallelism:
  - Thread-level parallelism (stamp out multiple pipelines)
  - Pipeline parallelism (create one long pipeline storing many packets at different stages)
- FPGA chips are now large SoCs (can run a control plane)



Programmable Hardware Alternatives

- FPGA Pros
  - Great performance with great programmability
  - With pipeline parallelism, can process line rate at 100Gbps+ with small packets in a reconfigurable pipeline
- FPGA Cons
  - Need to understand hardware arch and digital logic design – code in Verilog
  - Cost of logic per unit performance higher than ASIC (though lower than GP cores)
- Core-based NICs
  - Relatively easy to program in C, but performance is limited and parallelism is very course – difficult to scale up to 100Gbps+ without compromises
- ASIC-based NICs
  - Can implement efficient Match-Action Tables, but with limited flexibility
- Note: while packet-processing logic is cheaper on ASICs, on-chip memory / IO / DRAM / storage / SoC cores are not – total system cost is not necessarily that different

Programmable Hardware is coming to switches too!

- P4: a protocol-independent language for programming switches
- Very useful for inband monitoring, flow tracking, gateway scenarios
- Can we use this for Cloud SDN?

BAREFOOT NETWORKS



P4: Programming Protocol-Independent Packet Processors

Pat Bosshart[†], Dan Daly[*], Glen Gibb[†], Martin Izzard[†], Nick McKeown[‡], Jennifer Rexford[**], Cole Schlesinger[**], Dan Talayco[†], Amin Vahdat[¶], George Varghese[§], David Walker[**]
[†]Barefoot Networks [*]Intel [‡]Stanford University [**]Princeton University [¶]Google [§]Microsoft Research

ABSTRACT

P4 is a high-level language for programming protocol-independent packet processors. P4 works in conjunction with SDN control protocols like OpenFlow. In its current form, OpenFlow explicitly specifies protocol headers on which it operates. This set has grown from 12 to 41 fields in a few years, increasing the complexity of the specification while still not providing the flexibility to add new headers. In this paper we propose P4 as a strawman proposal for how OpenFlow should evolve in the future. We have three goals: (1) Reconfigurability in the field: Programmers should be able to change the way switches process packets once they are deployed. (2) Protocol independence: Switches should not be tied to any specific network protocols. (3) Target independence: Programmers should be able to describe packet-processing functionality independently of the specifics of the underlying hardware. As an example, we describe how to use P4 to configure a switch to add a new hierarchical label.

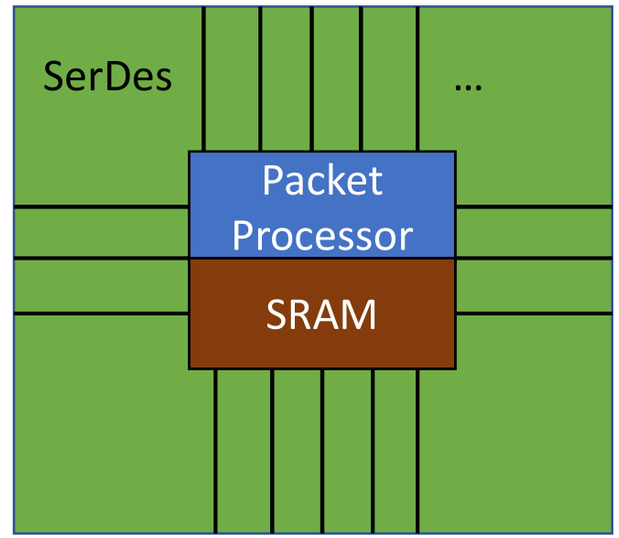
multiple stages of rule tables, to allow switches to expose more of their capabilities to the controller.

The proliferation of new header fields shows no signs of stopping. For example, data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality. Rather than repeatedly extending the OpenFlow specification, we argue that future switches should support flexible mechanisms for parsing packets and matching header fields, allowing controller applications to leverage these capabilities through a common, open interface (i.e., a new "OpenFlow 2.0" API). Such a general, extensible approach would be simpler, more elegant, and more future-proof than today's OpenFlow 1.x standard.

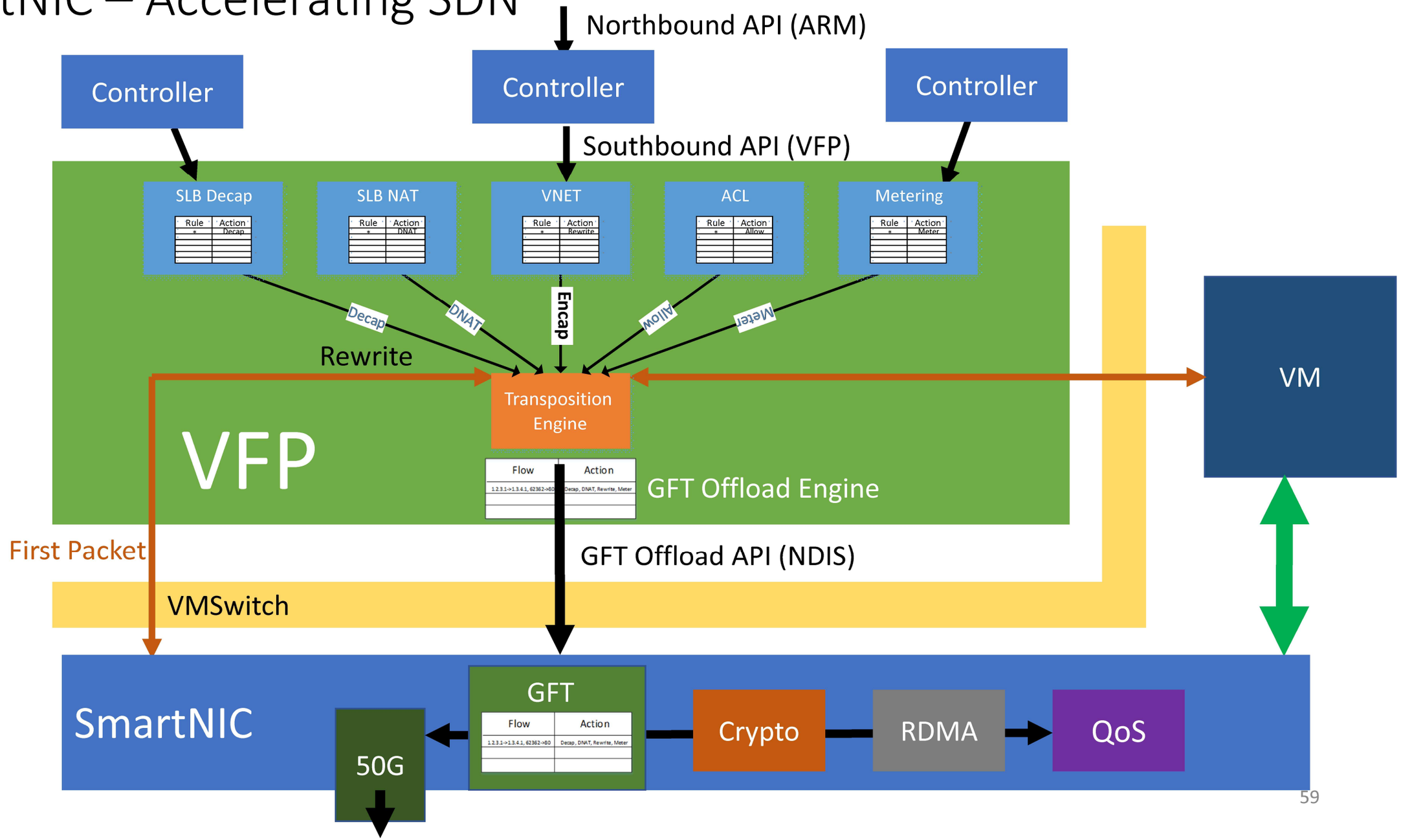


Limitations of Programmable Switching

- Programmable switching is great for protocol independence, telemetry, new monitoring insights, …
- Main Limits – flow scale (esp for per-connection policy)
- At 40-100Gbps on a host, can do flow lookups in DRAM - ~$10/GB
- Switches have to aggregate 30+ hosts – lookups require SRAM (2 orders of magnitude more expensive than DRAM)
- Hyperprogrammable switches have to compete with switches maxing out their I/O count (e.g. 32x400G, or 256 50G SerDes) – manufacturing limits dictate total space remaining for SRAM
- Lesson: Use programmability at the layer of the network where it makes sense for your application!



SmartNIC – Accelerating SDN

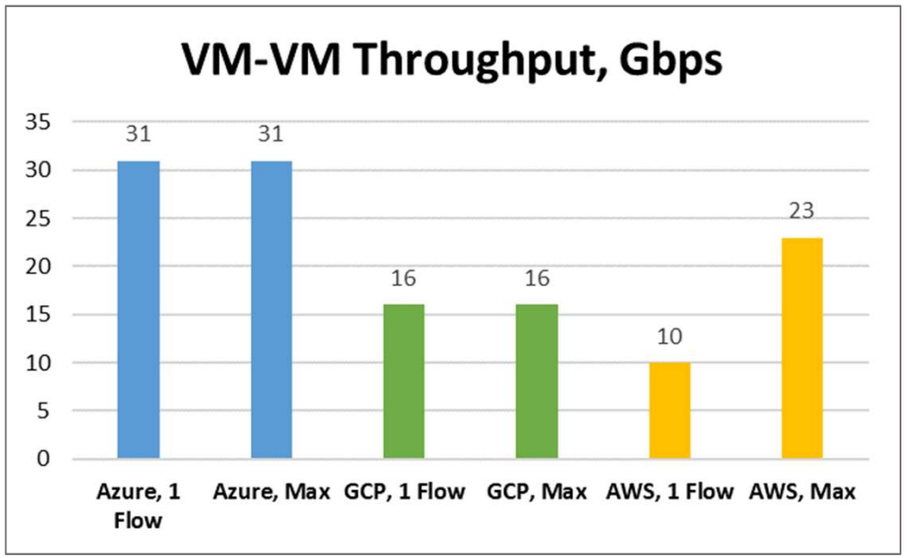


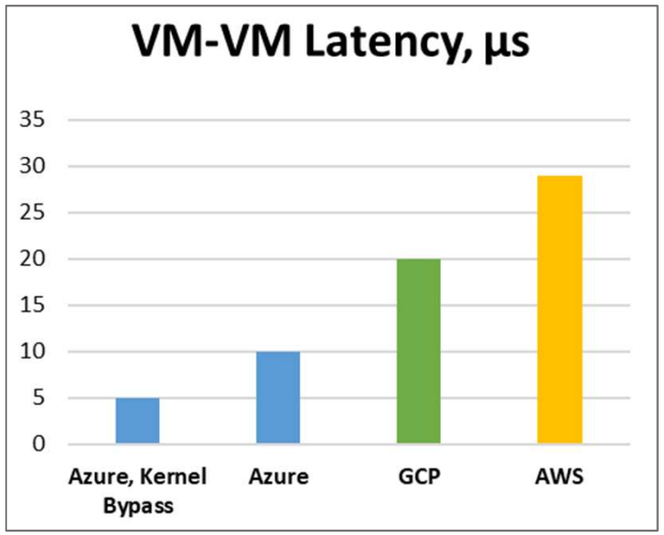
Azure Accelerated Networking

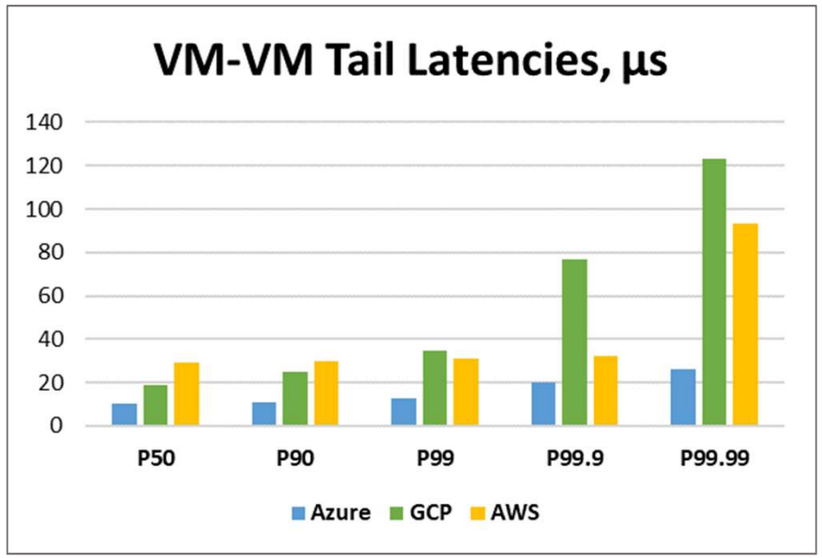


- Highest bandwidth VMs of any cloud so far…
  - Standard compute VMs get up to 32Gbps
  - Stock Linux VM with CUBIC gets 30+Gbps on a single connection
- Consistent low latency network performance
  - Provides SR-IOV to the VM
  - 5x+ latency improvement – sub 10us within tenants
  - Increased packets per second – Up to 25M PPS (12M forwarding) for DPDK VMs
  - Reduced jitter means more consistency in workloads
- Enables workloads requiring native performance to run in cloud VMs
  - >2x improvement for many DB and OLTP applications

AccelNet Comparative Results

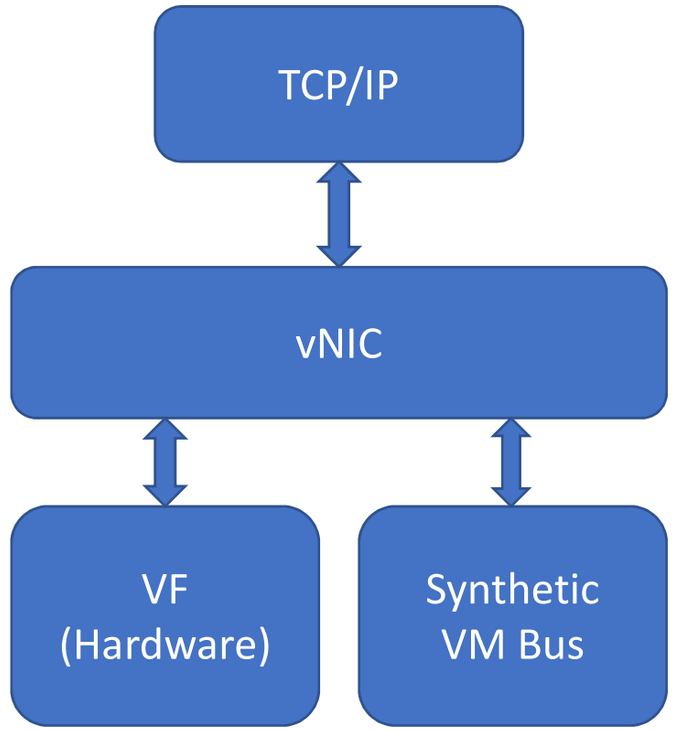






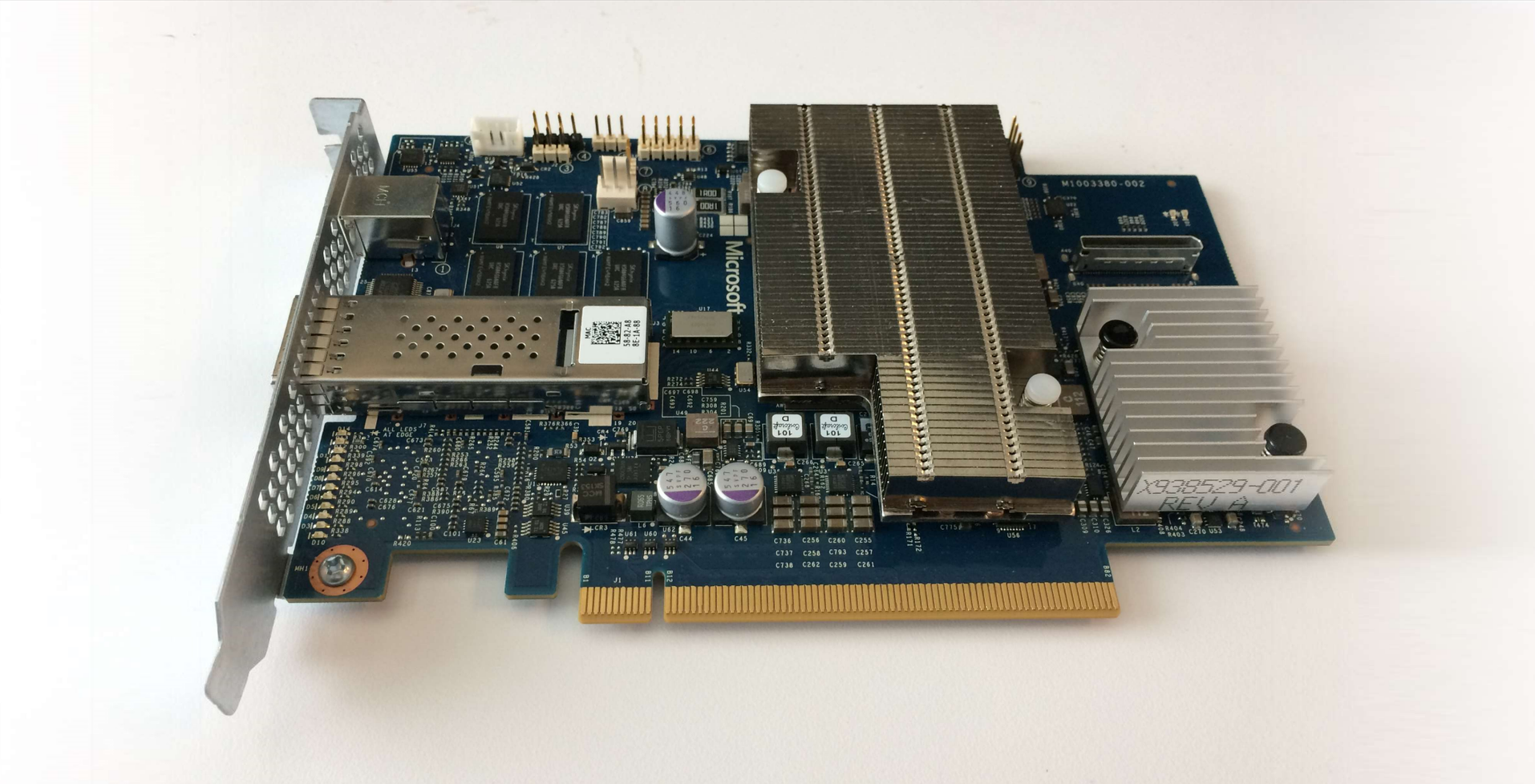
Serviceability is Key

- All parts of this system can be updated, any of which require us to take out the hardware path – or VM can be live migrated
  - FPGA image, driver, GFT layer, Vswitch/VFP, NIC PF driver
- IaaS requires high uptime and low disruption – can't take away the NIC device from under the app, and can't reboot the VM / app
- Instead, we keep the synthetic vNIC and support transparent failover between the vNIC and VF



Lesson: A huge amount of the effort to deploy SR-IOV was in making all parts of this path rebootlessly serviceable without impact

New **50GbE** SmartNIC for Project Olympus
(Announced at OCP 2017)



Changes, Changes, Changes

A few examples of many…

- TCP and protocol state machines
- Complex packet forwarding and duplication actions
- New SDN actions
- Accelerating the offload path
- Line rate diagnostics and monitoring

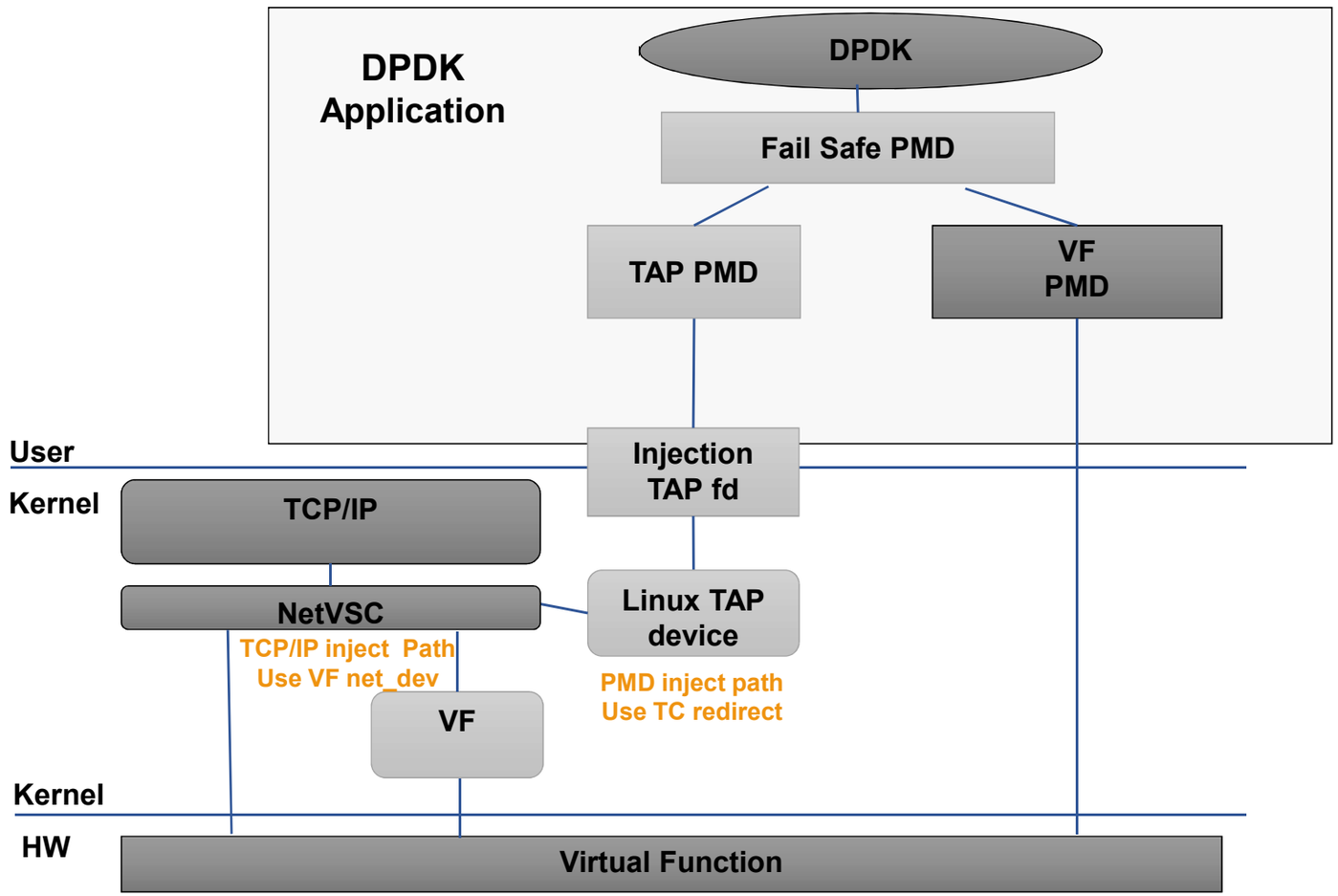
The Need for DPDK in the Cloud



- As customers migrate on-prem networks and apps to the cloud, virtualizing network functions (NFV) has become wildly popular
- Popular appliances in Azure Marketplace: Load Balancing, L7 filtering, web application firewalls, application gateways, DDoS protection, SD-WAN, and more
- DPDK, a popular framework for allowing userspace applications to poll the NIC directly, can improve performance, throughput, latency, and reliability for this important class of workloads
- Goal: How can we make DPDK appliances work well in the public cloud?
- Challenge: Serviceability

Azure Cloud DPDK with serviceability & migration support

- DPDK application binds over failsafe Polling Mode Driver
- Transparent failsafe works over VF PMD and Synthetic TAP PMD
- Principle: Warn the VM before an update to let it drain traffic, but don’t break the app
- Results: Up to 25M PPS (12M forwarding) for DPDK VMs on Azure, with full support for servicing and live migration in an application-transparent manner



Lessons Learned

- Design for serviceability upfront
- Use a unified development team
- Use software development techniques for FPGAs
- Better perf means better reliability
- HW/SW co-design is best when iterative
- Failure rates remained low – FPGAs in the DC were reasonably reliable
- Upper layers should be agnostic of offloads
- Mitigating Spectre performance impact

Looking Ahead

- Programmable hardware is offloading programmable software network functionality – what can we build in a hardware-native world?
- What could we do with a fully programmable hardware transport layer?
- Server perf is growing linearly, but cloud demand is exponential → all the pressure is on the network! What problems will we have to solve at the next 10x scale?





Interested in building the next 10x scale of networks? We have internships!
(And we're hiring!)

fstone@microsoft.com